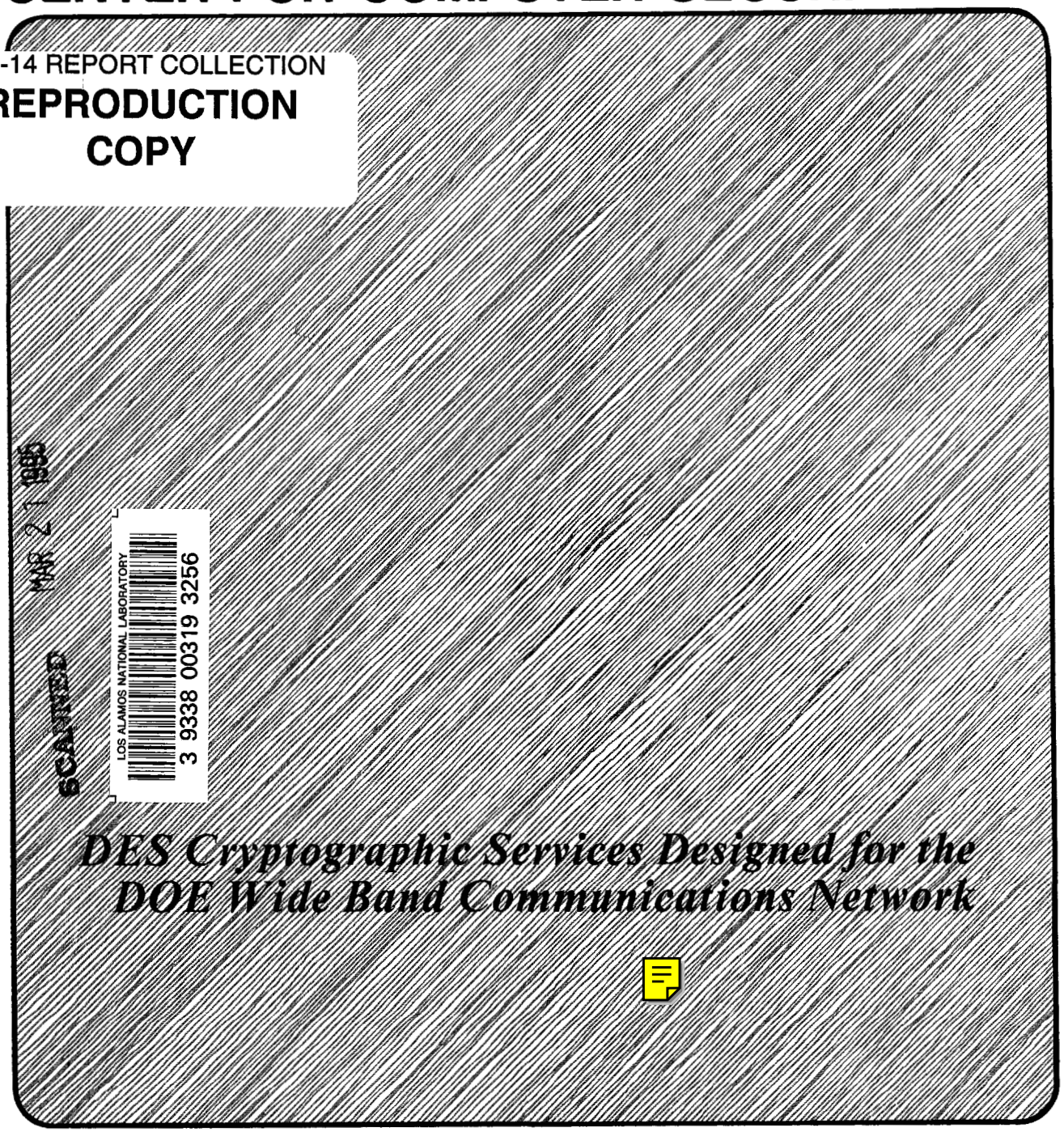
LA-10966-MS

c. 3

Department of Energy
CENTER FOR COMPUTER SECURITY



DES Cryptographic Services Designed for the DOE Wide Band Communications Network

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

This work was supported by the US Department of Energy, Office of Safeguards and Security, Computer and Technical Security Branch.

Prepared by Sharon Hurdle, Group N-4

LA-10966-MS

UC-32
Issued: November 1987

DES Cryptographic Services Designed for the DOE Wide Band Communications Network

Blaine Burnham

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

CONTENTS

WBCN Acronym List

CAD	Computer aided design
CAE	Computer aided engineering
CAM	Computer aided manufacturing
CIM	Computer integrated manufacturing
CM	Communication mechanism
CP	Cryptographic processing
DBMS	Database management system
DCL	DEC command language
DEC	Digital Equipment Corporation
DECnet	DEC networking software product
DES	Data Encryption Standard
INGRES	DEC database management system product
ISO	International Standards Organization
KEYINDEX	Key index table in KEYMGR database
KEYMGR	Key management database
KMM	Key management machine
LS	Link encryptor
N	Number of participating nodes
NBS	National Bureau of Standards
NK	Number of keys allocated per period
NODETABLE	Node name table in KEYMGR database
NP	Number of periods
NS	Network services
NSA	National Security Agency
NSP	Network Services Protocol
NWC	Nuclear Weapons Complex
OSI	Open systems interconnect
PRDTABLE	Period table in KEYMGR database
PTRTABLE	Pointer table in KEYMGR database
RN	Routing node
WBCN	Wideband Communications Network
WCP	WBCN communications protocol
WCS	WBCN cryptographic service

DES CRYPTOGRAPHIC SERVICES DESIGNED FOR THE DOE WIDE BAND COMMUNICATIONS NETWORK

by

Blaine Burnham

ABSTRACT

To exchange information between separate facilities of the Nuclear Weapons Complex quickly, accurately, and in computer-readable form, the Department of Energy (DOE) has built a secure data communications facility called the Wide Band Communications Network. The network proper achieves security by using high-grade encryption for the communications links and strict physical security measures at the nodes. To provide need-to-know separation among various elements and users of the network, data are encrypted before transmission using the National Bureau of Standards' Data Encryption Standard (DES) algorithm. The software described here provides the key management, key distribution, and cipher processing for this network.

I. INTRODUCTION

The Nuclear Weapons Complex (NWC)/Wide Band Communications Network (WBCN) is a wide area network being implemented to enable, at least initially, the integration of NWC CAD/CAM/CAE and manufacturing functions as computer integrated manufacturing (CIM). This network is intended to provide the connectivity among the NWC facilities required by the CIM project.

Each node on the WBCN will use a Digital Equipment Corporation (DEC) VAX 11/750 computer as a dedicated gateway. These gateways will be the sole entry point for each facility to the network. The WBCN will use the DECnet communications protocol between the gateways. Because of the historic autonomy of the Department of Energy (DOE) contractors, a variety of mechanisms will be used to communicate between the dedicated gateway and the local information processing facilities. Each gateway will operate in a protected environment accessed only by Q-cleared or escorted personnel. The gateways will not contain any user accounts and will execute only dedicated processes.

The communications links between gateways will be protected by the appropriate link encryption. However, as a limitation of link encryption, any information that traverses a gateway will do so in a cleartext form.

A few of these gateway machines are so positioned that they also serve as routing nodes. This means that information not destined for the facility at that gateway must pass through the gateway. The WBCN cryptographic service (WCS) is designed to provide a mechanism for preserving the confidentiality of the routed information as it traverses the gateways of intermediate facilities.

Like any other cryptographic implementation, the WCS has had to deal with the problem of key distribution and coordination as well as providing the cryptographic services. The remainder of this report describes the details of the problem and in doing so outlines the constraints and requirements for the solution. This description is followed by a detailed discussion of the mechanism that has been developed to address the problem. Appended to this report are a description of the data structure (Appendix A), a listing of all the developed software (Appendix B), and a users guide (Appendix C).

II. THE PROBLEM

The establishment of the WBCN within the DOE has brought forth a well-recognized difficulty with multiuser switched communications networks. That difficulty is how to maintain the confidentiality of a message exchanged between two participants when the data path is exposed, to some degree, to an intermediate third party.

The simplest case is that of just two legitimate participants (users A and B) and a communications mechanism connecting them (Fig. 1). In this configuration there is no problem. Anything that A sends to B can go nowhere else and vice versa. The situation becomes more complicated if there is an intruder (D) who attempts to intercept the information on the communications medium (Fig. 2). The efforts of D can be effectively thwarted by employing various techniques to harden the communications medium. These techniques include a variety of physical alterations, for example, routing the communications medium through a protected distribution system, thereby denying access or considerably reducing the apparent information content of the signal. For example, A or B might apply cryptographic processing to the information before it enters the communications medium (Fig. 3).

Without access to the protected cryptographic variable, the information content of the signal will appear to be worthless to D, thus maintaining the confidentiality of the message shared by A and B. This form of cryptographic protection is called link encryption because the communications link between A and B is protected by the cryptographic application. It is important to note that link encryption affects all data on the link.

The situation becomes yet more complicated if C is a legitimate user of the communications medium and has need to exchange information with the other participants. Such a configuration is shown in Fig. 4. In this configuration, a participant (node) can support a connection with a single other node and thwart the efforts of an external intruder (D) to capture information from the communications media. However, if the cryptographic variable information (key) is available to all the nodes, there can be no private two-party communications because the third party would be able to eavesdrop on the communications media.

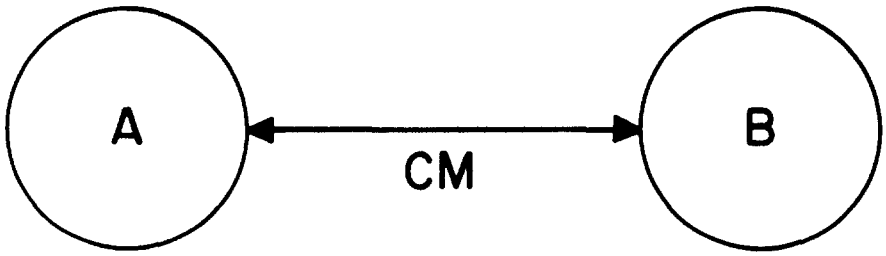


Fig. 1. The simplest case is that of just two legitimate participants (users A and B) and a communications mechanism (CM) connecting them.

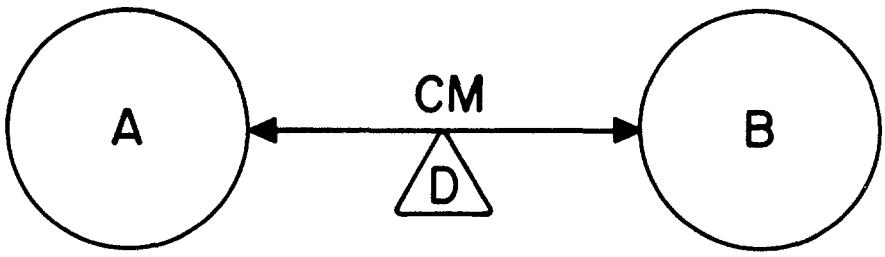


Fig. 2. The situation becomes more complicated if there is an intruder (D) who attempts to intercept the information on the communications medium.

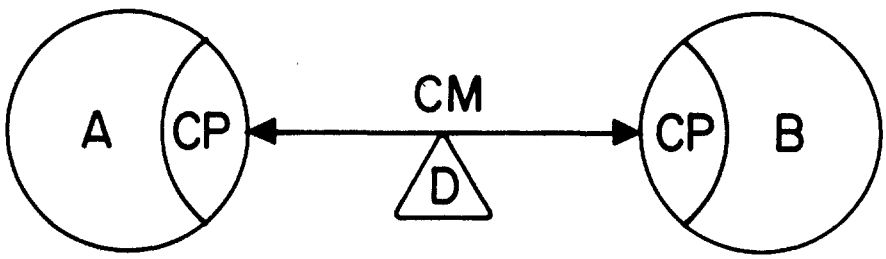


Fig. 3. These techniques include a variety of physical alterations, for example, routing the communications medium through a protected distribution system, thereby denying access to the medium, or considerably reducing the apparent information content of the signal on the communications medium, for example, by applying cryptographic processing (CP) to the information before it enters the communications medium.

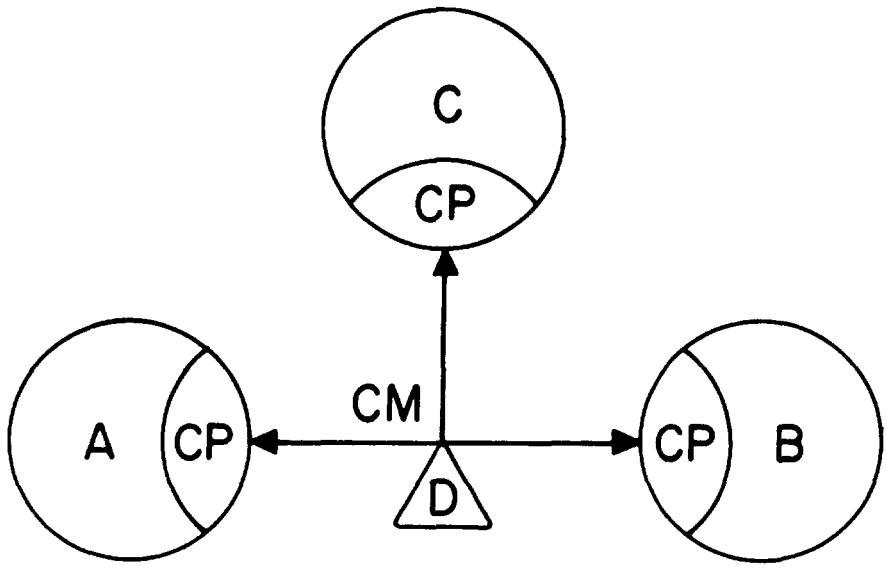


Fig. 4. The situation becomes yet more complicated if C is a legitimate user of the communications medium and has need to exchange information with the other participants.

The requirement to provide confidential two-party communications among the three nodes in Fig. 4 can be met by using creative key management. In particular, if the keys shared by A and B are unavailable to C, the requirement is met. The most significant shortcoming of the Fig. 4 configuration is the inability to support concurrent connections among all the nodes.

The requirement that each node be able to maintain concurrent communications with each of the other nodes results in a connection configuration much like that shown in Fig. 5. This configuration does not require the complicated level of key management posed by the model shown in Fig. 4; furthermore, it supports concurrent connections with each of the participants. This configuration does require the replication of crypto devices for each of the participants. If there are n nodes, then $(n^2 - n)$ devices and $(n^2 - n)/2$ communications links are required. However, such a configuration provides a secure, attainable, and to some extent practical solution to the problem of providing two-party confidentiality among the nodes. Its shortcoming is that the cost increases as the square of the number of nodes.

The link encryption solution fails if the requirements are extended to include a routing node in the communications path as shown in Fig. 6. There is no link encryption solution that will provide confidential two-party communications in this configuration. This configuration is very typical of packet switched multinode communications networks. In such a network, it is frequently the case that not every node has a separate, unique connection to every other node in the network. Among other things, this would require $(n^2 - n)/2$ connections. The network solution to reducing the required number of connections is to provide a mechanism that allows a message originating at A and destined for B to be routed through C. This mechanism represents an enormous increase in sophistication of the network over the previous models.

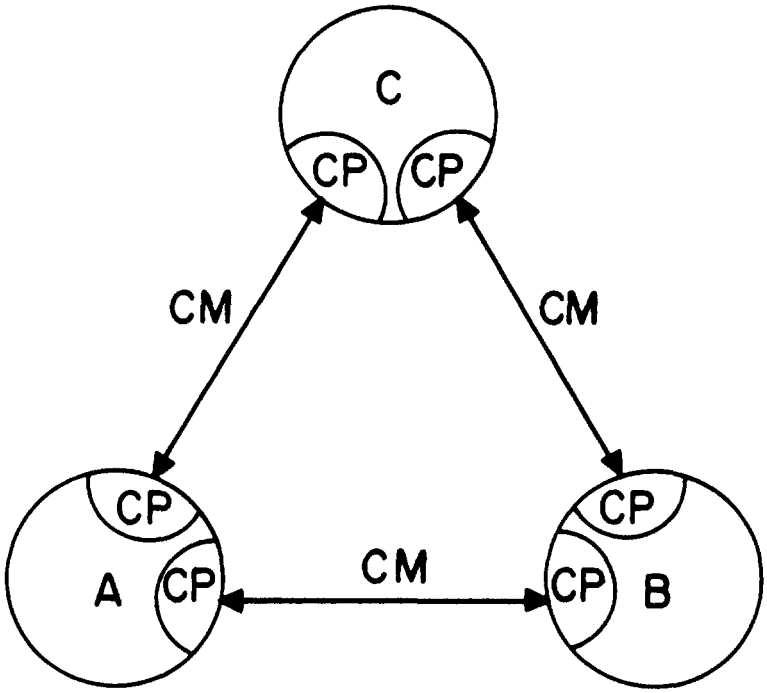


Fig. 5. The additional requirement that each node be able to maintain concurrent communications with each of the other nodes results in a connection configuration.

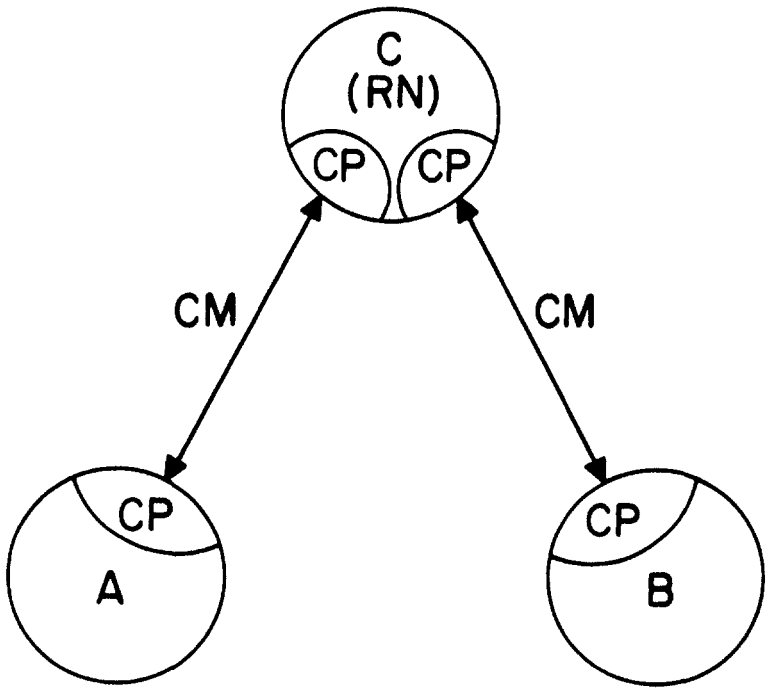


Fig. 6. The link encryption solution fails if the requirements are extended to include a routing node (RN) in the communications path.

The basic element required to implement this mechanism is the ability to append an "addressing label" to the message sent by A to B such that C can interrogate this label, decide to accept or reject the message (that is, determine that the message is not being sent to C), and then forward the message to B. The critical part of the mechanism is that the message has a label and any node can read it. For that reason, link encryption will no longer provide the required pairwise confidentiality between A and B. If link encryption is interposed from A to B, then C cannot read any of the message, in particular the label. Hence, C would be unable to forward (route) the message on to B. If the link encryption is imposed from A to C and then from C to B, C is able to violate the A to B confidentiality because the entire message is in cleartext at C. The configuration shown in Fig. 6 is typical of the WBCN. The solution to preserving two-party confidentiality in a network that includes routing is to provide for the encryption of the message while leaving the label unencrypted. In this way, the routing node has access to the necessary information without being able to read the message. There are two basically different approaches that accomplish this end.

One approach to meeting the requirement is to endow the link encryptor with considerable sophistication to enable it to read the entire data transmission stream; distinguish between the label and the message; cipher-process the message, in place; and transmit the message with the cleartext label over the network.

Another approach intercepts the message in the process of being labeled, encipher-processes the message at interception, and proceeds with the labeling and transmission. In either case, there are nontrivial problems of synchronizing the encipherment processing and providing for key distribution.

The approach, called embedded encryption, permits a variety of techniques of application. This is the approach used to provide for the WBCN cryptographic services. In order to provide a more complete understanding of this approach to message protection, there is another aspect of communications that needs to be introduced--that is, the notion of communications protocols. A communications protocol is a procedure for communicating between two or more nodes on a network. This procedure is usually subdivided into functionally specific subsets or layers. Each layer implements a specific collection of tasks that identify and label the message and ensure the correct, accurate delivery of the message over the network. The lowest layer of a protocol deals with the physical communications requirement. The most abstract layer provides an interface to the various user-created or -invoked processes operating on the application machine. The International Standards Organization has developed a seven-layer model of communications protocols. This is the open system interconnect standard, ISO/OSI.[1] DECnet,[2] the protocol used on the WBCN, implements most of the functions of this standard. DECnet has a five-layer architecture and is moving toward alignment with the ISO/OSI seven-layer architecture.

The embedded encryption method of providing confidentiality on the network requires that the cipher processing be implemented somewhere in the DECnet protocol before the layer that creates and checks the routing information. This implementation results in cryptographic protection of

the message while leaving the routing information available for interpretation by the routing processes. For the WBCN requirements, the most desirable implementation would be in the Network Services Protocol (NSP) layer of DECnet. Such an implementation would be transparent to the WBCN yet provide the required message confidentiality.

A prototype of such implementation has been created. This NSP implementation required a considerable modification of the standard DECnet and the development of custom hardware. Although this prototype was the most desirable solution, it was not acceptable because the modified DECnet was not supported by a vendor and could not be supported within the constraints of the WBCN project. The remaining alternative has been to do the cipher implementation at the user protocol level. This is the option used for the WBCN cipher services. These services consist of three distinct elements: centralized key distribution, node-specific key management, and encryption/decryption service utilities. A schematic of the generic WBCN crypto services is shown in Fig. 7.

In this configuration, the network processing for each participant is contained within a secure environment (dotted line). The information in the communication links is protected by the link encryptors imposed on all

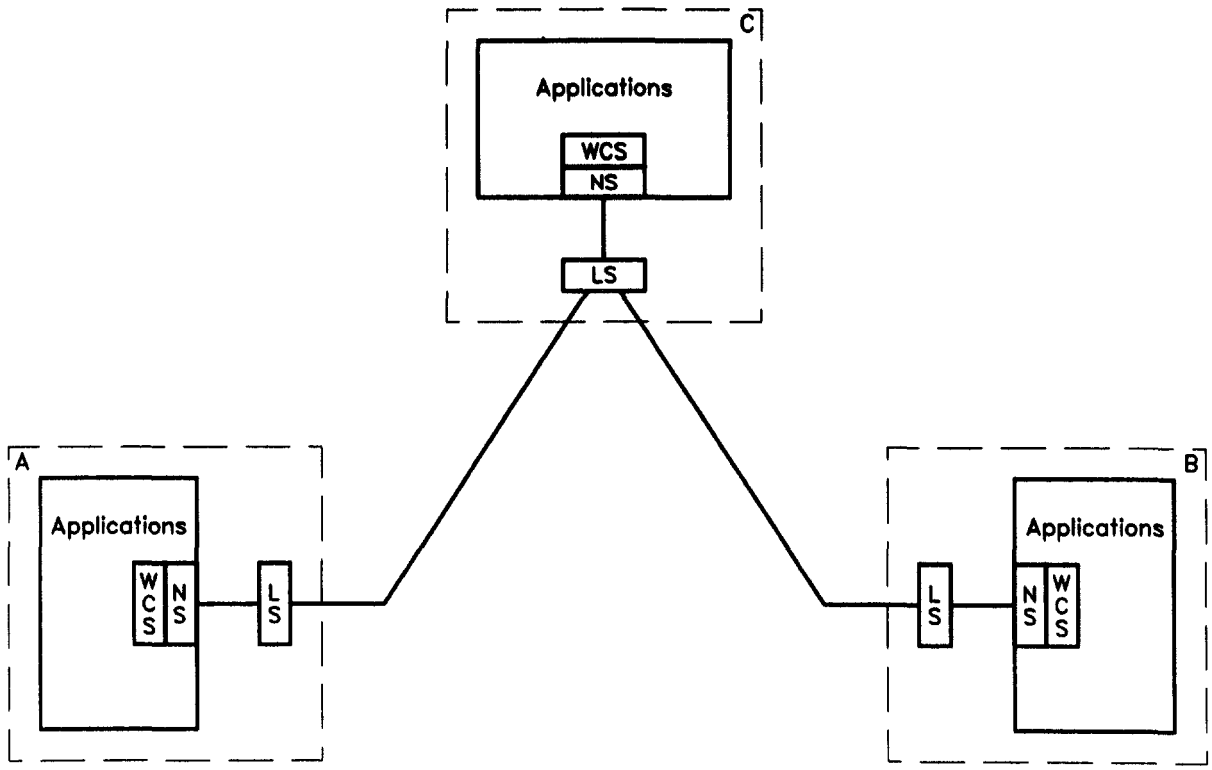


Fig. 7. A schematic of the generic WBCN crypto services (WCS) with the network communications services (NS) and link encryptors (LS).

lines. The WCS is available to the application level processes for utilization by the WBCN communications protocol (WCP) that is being developed to enable the various WBCN functions (for example, file push). The WCP does the cipher message processing before invoking the network communications services. In this fashion the network processing is not interrupted. The WCS is composed of three distinct elements, two of which are located on each node in the network; the third runs on a single dedicated machine.

The single dedicated element is the centralized key management. The two elements shared by each node are the node-specific key management and the cipher processes.

III. KEY MANAGEMENT

The key management structure and implementation form the foundation for the success or failure of any cipher-processing service. There is a tendency for key management schemes to become very complex very quickly. This tendency must be resisted at every turn. The key management scheme for the WCS is the critical element that actually enforces the required message confidentiality. The key management extracts, from a pool of available keys, mutually distinct sets of keys, a set for each node. Only the keys that that node will use to communicate with any other node are in its set. However, the node receives none of the keys used between any other pair of nodes. This key partitioning is illustrated in Fig. 8.

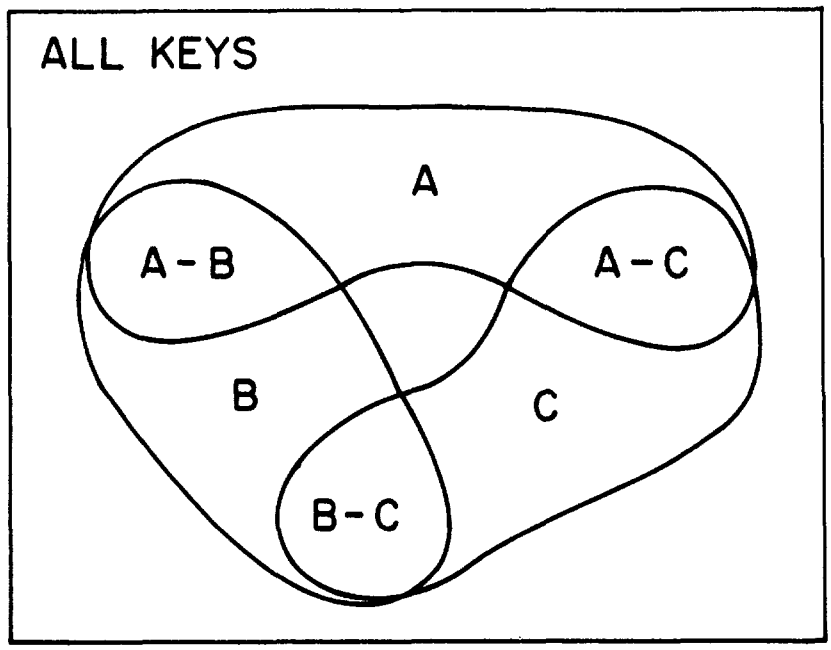


Fig. 8. The key management extracts, from a pool of available keys, mutually distinct sets of keys, a set for each node. Only the keys that that node will use to communicate with any other node are in that set. However, the node receives none of the keys used between any other pair of nodes.

In this example, there are three nodes, each with the requirement to have confidential communications with each of the other two nodes. The keys labeled A-B are those to be used between nodes A and B; A-C are those used between nodes and A and C; and B-C are those used between B and C. Under the WBCN key management implementation, node A received key sets A-B and A-C; node B received key sets A-B and B-C; and node C received key sets B-C and A-C. Observe that with this scheme nodes B and C share keys that node A does not possess. Thus, B and C can exchange encrypted messages that A cannot decipher for lack of the key.

There are a number of constraints that have contributed to the design of the implementation of this scheme. Any Data Encryption Standard (DES) application within the DOE currently is required to utilize cryptographic variables approved or supplied by the National Security Agency (NSA). In the case of the WCS, an agreement was negotiated with NSA for them to supply DOE with the keys. The keys are delivered, in bulk, on magnetic tape.

The WCP must be able to synchronize the key variable application. The two participants must have some means of identifying which key to apply to a particular message. The keys must have some form of frequency-of-use management so that the same key is not used with many consecutive messages. The key management must be simple and robust enough to withstand operation by an ever-changing operation staff. From a practical point of view, it was apparent that this key management would have many of the same requirements as a conventional data management task. The key management element has been developed with these constraints in mind. An initial decision was made to use a single database management system (DBMS) product to facilitate the whole WCS. INGRES was the DBMS chosen for the WCS. The key management was achieved primarily with DIGITAL command language (DCL) command files, FORTRAN programs, and INGRES scripts.

In part, these constraints determine the steps of the logic and information flow of the key management scheme. These steps are very closely matched by the components of the actual software as given in Appendix B.

IV. THE CENTRALIZED KEY MANAGEMENT

An environment for the key management must be created. The first step is the creation of the actual database on the key management machine (KMM). This step is accomplished by the DCL command procedure SETUP.COM. This DCL command file creates the key management database (KEYMGR) and in that database the pointer table (PTRTABLE), the key index table (KEYINDEX), the period table (PRDTABLE), and the node name table (NODETABLE). The next step is the loading of the NSA keys off the magnetic tape into the database on the KMM. This loading is accomplished with the KEYLOAD.COM command procedure. In addition, this process affixes the permanent pointer to the newly loaded keys. It is this pointer that is exchanged among nodes to indicate what key is being used to cipher-process a given message. This affixing process is performed by the FORTRAN program INDEX. The key and pointer pairs are then appended to the DBMS table KEYINDEX, which is the master key table and contains all the keys. This same program, INDEX, sets and adjusts the pointers in PTRTABLE. These pointers are described in detail in Appendix A. With the completion of KEYLOAD.COM, all the NSA keys are indexed and stored in the master key table awaiting distribution.

The key management scheme needs to have a record of all the participating nodes. This record is used to determine the distribution of keys and how many keys are needed. The assumption has been made that communication requirements are equally likely among any pair of nodes so no attempt has been made to weigh the node-specific key allocation in favor of the high-traffic pairs. The DCL command file UPDATE.COM provides the operator of the KMM with an updated data table NODETABLE. This table (see Appendix A) includes the node management data items as well as the node identifier. The node identifier used in the node table must be the same as the response to the DCL command $ SHOW LOGICAL SYSNODE for each of the WBCN gateway nodes. This identification is checked at a later step in the key distribution process.

The next consideration in the key management is the key period, which is the length of time a given key is active. Key periods are traditionally determined as a function of the relevance of the protected information, the life expectancy of the protected message, the volume of messages, the redundancy within the message, and the level of hostility of the environment. The WBCN application of DES is unique in that it is being used to provide a need-to-know information confidentiality rather than information protection. The protection is provided by the approved link encryption processors used on all the communication circuits. As a result, the WCS key period management requirements are less stringent. On the other hand, the WBCN Security Committee decided against applying a single key, for some period, to all the messages between two particular nodes. They concluded that a better scheme would be to have a pool of keys per period to be used between nodes. In this fashion, it would be very unlikely that consecutive messages would be encrypted under the same key. The upshot of all of this is the key management scheme sets the key period to be 1 month and allows for adjusting the number of keys allocated to the key pool. The default value is 20 keys per pool per period. This value is one of the attributes in the data table PTRTABLE. The last wrinkle in the consideration of key periods is how many periods' worth of key would be distributed and stored on the participating nodes at any time. Again the WBCN Security Committee settled on three periods at one time.

Most of the key period information is handled automatically by the key management processes. However, because it depends on the internal calendar of the KMM, it must be initialized. This initialization is accomplished with the DCL command procedure PERIOD.COM.

Having completed the above three steps, the KMM is prepared to create the separate key sets for each node. This creation is accomplished with the DCL procedure EXTRACT.COM. This process determines from the NODETABLE entries which key sets must be created; determines the number of keys needed for distribution [note that it requires $(N^2 - N)*NP*NK$ keys per key distribution where N is the number of participating nodes, NP is the number of periods, and NK is the number of keys allocated per period]; checks the number of keys needed against the number available; and builds VMS* files of the distinct key sets identified by origination node, destination node, key pointer period, and key. This process also deletes from the master key table KEYINDEX all the keys extracted for this distribution and resets the pointer in the PTRTABLE to reflect the distribution. This

*Virtual Memory System (VMS) is the name of the operating system supplied by Digital Equipment Corporation for its VAX line of computers.

deletion is a protection against having the master table compromised. At the conclusion of EXTRACT.COM, all the distinct key sets are stored as VMS files on the KMM.

The next step is the creation of the actual key distribution tapes to be sent to each node. This creation is accomplished by the DCL procedure DISTRIB.COM. This process actually writes the key set files on tape for distribution. The key set files are identified by incorporating the node name information in the file name. This DCL procedure captures the information and prompts the operator to be very conscientious in identifying the destination node name with the information written on the external label affixed to the tape reel. It is essential that the correct key set be sent to the correct node. Substantial attention is paid to helping the operator through this critical action.

The DISTRIB.COM procedure performs its own garbage collection as it writes the VMS files out to the tape volumes. However, as a precaution, the current VMS key files are copied to a set of archive files. This precaution is taken because if something goes wrong with the tape distribution to one of the nodes, there would no way to recover the key set that should have gone to it. The loss of key sets during distribution will functionally incapacitate the miskeyed nodes. A permutation of key sets will keep the intended recipient from receiving his designated information although a third party with the misdirected key set would be able to decipher and recover the information.

The archived key sets remain on the KMM until the next subsequent key distribution occurs with the invocation of EXTRACT.COM. This process deletes any archived key sets before it prepares the current new key sets.

DISTRIB.COM is the last of the key distribution procedures that run on the KMM. At the completion of DISTRIB.COM, all the key sets have been written to tape and are ready to be distributed to the respective nodes.

In general, the key distribution processes, with two exceptions, are asynchronous; that is, they can be run in any order. The first exception is SETUP.COM. As was described above, this command procedure establishes the KEYMGR. SETUP.COM must be run successfully once to set up the database on the KMM. It is frequently the case that a machine and its accounts have not been set up correctly for the creation of a database. As a result, SETUP.COM may not run successfully the first time or the second or even the third time it is tried. Partial execution of SETUP.COM may leave a lot of unusable database structure in place so the first thing SETUP.COM does is to issue a DESTROY KEYMGR command. This action does all the necessary garbage removal prior to creating the database. The important point is DESTROY KEYMGR will completely eliminate a perfectly functional database as well as clean up a bad database. If SETUP.COM is run after the database has been created and loaded with information, all the contents will be gone. I strongly recommend that SETUP.COM be run successfully once and then removed from the KMM to preclude the possibility of subsequent damage.

Clearly, keys must be loaded on the database before they can be distributed. However, the keyload process can be run any time key information is available. It is not necessary to synchronize it with any of the other processes.

The creation of key sets with EXTRACT.COM depends only on having enough keys available. EXTRACT.COM checks for enough keys, issues an error

message if there are not enough keys in the master table, and then exits with no further action. However, EXTRACT.COM will have deleted the archive key files prior to the error message.

It does not make sense to run the DISTRIB.COM process without having prepared the key files using EXTRACT.COM first. However, DISTRIB.COM checks for key files and issues a warning message before it exits if none are found.

All these individual functional processes have been incorporated into a single, menu-driven executive procedure, ROOT.COM. This process incorporates menu-selected branches to all the described individual processes with the exception that ROOT.COM cannot invoke SETUP.COM. ROOT.COM has been designed to be the operator interface to the key management process set. It is self-guiding and highly dialogue oriented.

The completion of the command procedure DISTRIB.COM on the KMM prepares key set tables to be distributed to the WBCN nodes. It also is the end of the activity of the KMM in the WCS.

V. NODE-SPECIFIC KEY MANAGEMENT

The remaining two elements of WCS are the node-specific key management and the cipher processes. The node-specific key management consists of two command procedures: SETUP.COM and KEYLOAD.COM. SETUP.COM is the DCL command procedure that creates the INGRES key user database on the node machine. Like the SETUP.COM procedure on the KMM, this should be run successfully once and then removed from the system for the same reason.

After the database has been created, that is, SETUP.COM runs successfully, the next step is the loading of the tape prepared on the KMM and distributed to the node site. This loading is accomplished by the KEYLOAD.COM process. Much like the KMM processes, KEYLOAD.COM is a self-guided, dialogue-oriented procedure. All that is needed from the operator is mounting of the distribution tape.

VI. THE CIPHER PROCESS

The last component of the WCS is the cipher process. The cipher process designed for the WBCN is a DES based cipher-block-chaining implementation. The process is accessed by making a call to the subroutine cbc_cipher (see listing in Appendix B). This subroutine provides the interface to the VMS assembly language implementation of the DES algorithm written by Rich Belles at Lawrence Livermore National Laboratory. This software algorithm has been checked and verified against the National Bureau of Standards publication DES Modes of Operation.[3]

The interface is set up to encrypt or decrypt an integral number of 64-bit words. The arguments required by this subroutine are detailed in the listing of the code (Appendix B).

There is a caution that must be observed by the users of this WCS. The cipher processor anticipates the cipher text or cleartext to be integral numbers of 64-bit words, quad words. The last word in the message stream must be padded out to the quad word boundary. The DES algorithm receives the starting address of the text and the number of quad words.

The algorithm takes full quad words whether they are padded or not. It is probably better for the WSP to do the padding rather than to let the algorithm pluck the random bits that fill out the quad word.

VII. SUMMARY

The WBCN is an initial step in establishing a full functionality digital packet-switched network for the DOE. This network is national in scope and continental in dimension. One of the many problems encountered in the implementation is providing a high level of assurance that the network would maintain the confidentiality of two-party communications. To that end, a number of alternative possibilities have been investigated. This inquiry has led to the decision to develop an application level data encryption service to be utilized by the developers of the WCP. This service, the WBCN cryptographic service, consists of multiple elements that provide both key management and cipher processing. The key management elements incorporate many aspects of traditional data management and have been developed using the INGRES DBMS.

Mechanisms have been incorporated into the key management elements to provide for key information exchange between nodes without the necessity of developing an elaborate session key, key-encrypting key, or super-key protocol.

The cipher-processing element is based on a verified VMS assembly language implementation of the DES algorithm.

The entire WBCN cipher services suite has been loaded on the key management machine and executed correctly. Key distribution from NSA has been negotiated. The first distribution of keys has been accomplished and the working keys have been loaded on the KMM.

REFERENCES

1. DECnet, DIGITAL Network Architecture (Phase IV), General Description, Order #AA-N149A-TC (Digital Equipment Corporation, Maynard, MA).

2. Wendy B. Rauch-Hindin, "Upper-Level Network Protocols," Electronic Design (March 3, 1983), pp. 180-194.

3. National Bureau of Standards, DES Modes of Operation, Federal Information Processing Standard (FIPS) publication No. 81 (December 20, 1980).

APPENDIX A

DESCRIPTION OF THE DATA STRUCTURE

This appendix describes the data tables implemented under the INGRES DBMS for the key management in the WBCN cipher services. The key management is basically a data management problem. The INGRES DBMS has been used to implement the data management part of the key management. INGRES is a relational DBMS that uses flat files (tables) as its logical data organizational structure. This appendix describes the tables used in the various elements of the key management implementation. Some of the tables are scratch tables as opposed to the permanent tables. Permanent tables contain information that transcends any instance of specific key distribution. The scratch tables are distribution specific and are created and destroyed during a single distribution cycle. The tables are identified with which process creates them or destroys them and their contents.

Element: Centralized key management

Table: PTR TABLE

P/S: Permanent

Created: SETUP.COM

Table Description: This table contains the pointer into the KEYINDEX table and the period tracking data.

Column Definition:
ATTR = C10
PTR = I4

Row Entries & Initial Values:

ATTR	PTR
HIGHPTR	1
LOWPTR	1
PRDPTR	1
NBRKEYS	20
NBRPRDS	3
NEXTPRD	1

Entry description: HIGHPTR: The PTR value associated with this ATTR indicates the next sequential index that is free to be assigned to a key that is received from NSA distribution. The associated PTR is initialized to 1 when the table is created because no keys have been received, indexed, and stored in the KEYINDEX TABLE.

LOWPTR: The PTR value associated with this ATTR indicates the next sequential index of a key that is available for distribution to the nodes.

The difference: PTR (HIGHPTR) - PTR (LOWPTR) is the total number of keys stored in the wastes table KEYINDEX. This difference is checked against the number of keys required for a particular distribution in the program EXTRACT.EXE under the control of EXTRACT.COM. If the difference is too small, a warning message is issued and the key set preparation is stopped. If the difference is large enough, the key set preparation runs to completion. All the required keys are copied from the KEYINDEX table into VMS files. Then PTR (LOWPTR) is recalculated to reflect the drawdown from the master key table and all keys with indexes less than the new PTR (LOWPTR) are deleted from the master table.

PRDPTR: The PTR value associated with this ATTR indicates which key period was last assigned.

NBRKEYS: The PTR value associated with this ATTR is the number of keys each period to be associated with a node to communicate with a specific other node. This PTR value is set at 20. If the nodes are A and B with A being the origination node, then A has 20 distinct keys to use when originating messages to B. Likewise, B will have 20 different keys to use when originating messages to A. In all, A and B share those 40 message keys for that period.

NBRPRDS: The PTR value associated with this value of ATTR is the number of periods' worth of keys per distribution. This value is initialized at 3. Thus, as a single distribution tape to A, there will be 60 distinct keys subdivided into three sets of 20 each for A to use to originate messages to B.

NEXTPRD: The PTR value associated with this ATTR is the number of the next assignable key period. For example, after the first distribution, PTR(NEXTPRD) = 4.

Element:	Centralized key management
Table:	KEYINDEX
P/S:	Permanent
Created:	SETUP.COM
Table Description:	This is the master key table.
Column Definition:	PTR = I4 KEY = C16
Row Entries:	Each row is a key pointer pair, one for each NSA-distributed key.
Entry Description:	The pointer is the index associated with each new incoming key by the program INDEX.EXE under the control of KEYLOAD.COM. For example, if 1253 keys had been initially loaded and no keys distributed, PTR(LOWPTR) = 1, PTR(HIGHPTR) = 1254. The keys stored in KEYINDEX would have pointer values ranging from 1-1253. If another 100 new keys were to be loaded, the first of these keys would be assigned pointer value PTR(HIGHPTR) = 1254. The last key would be assigned value 1353 and at the conclusion of the loading, PTR(HIGHPTR) = 1354.

Element:	Centralized key distribution
Table:	PRDTABLE
P/S:	Permanent
Created:	SETUP.COM
Table Description:	This table keeps track of the key periods and the data when the period expires.
Column Definition:	PERIOD = I4 PERIOD_DATE = date
Row Entries:	Each row is a period value and expiration date of the period.
Entry Description:	In order for each node to keep track of which keys to use, the node must know which key period to reference. I chose to distribute the period information rather than hardwire it into the code. I did this because I am not sure how the key period management may eventually mature. The way it is set, the global period management can be adjusted through change to this one table. Because the reference date associated with each period is an absolute date rather than an increment from a starting date, the table must be initialized at the onset of the utilization of the WCS. This table is accessed through the DCL command PERIOD.COM under the control of ROOT.COM.

Element: Centralized key management

Table: NODETABLE

P/S: Permanent

Created: SETUP.COM

Table Description: This table contains the WBCN node identification used to identify the nodes with the distributed keysets. In addition, the table includes the general demographic information about node site location, configuration, and contacts for the benefit of centralized WBCN management.

Column Definition:

NODE_NAME	= C20
SITE_NAME	= C20
DECNET_ADDR	= C20
MAIL_ADDR1	= C20
MAIL_ADDR2	= C20
MAIL_ADDR3	= C20
MAIL_ADDR4	= C20
NAME_SYSNGR	= C20
NAME_SYSNGR	= C20
NAME_DCOM	= C20
FPH_SYSMGR	= C12
CPH_SYSMGR	= C12
FPH_SECMGR	= C12
CPH_SECMGR	= C12
FPH_DCOM	= C12
CPH_DCOM	= C12
CONFIG_INFO	= C200
LOCNET_INFO	= C200

Row Entries: Each row is the complete WBCN node identification entry in the WCS key management database.

Entry Description: NODE_NAME: This entry stores the unique WBCN node identifier for each node and is the identifier used to prepare and distinguish the key sets. It is required that the contents of this identifier be exactly the same as those shown in response to the DCL command $ SHOW LOGICAL SYSNODE on each WBCN node. I have used this symbol to check for valid key distribution at the separate nodes.

The remainder of the record attributes are site-specific demographic information. This information is not used directly in the WCS.

SITE_NAME: The common value of the facility housing/ using the WBCN gateway.

DECNET_ADDR: The DECNET node designation.

MAIL_ADDR(1-4): The mailing address--four lines available.

NAME_SYSMGR: The name of the system manager.

NAME__SECMGR: The name of the WBCN node security manager.

NAME__DCOM: The name of the digital communications contact.

FPH_****: The FTS phone numbers for the named contacts.

CPH_****: The commercial phone numbers for the named contacts.

CONFIG__INFOR: Optional space to keep track of the WBCN node machine configuration.

LOCNET_INFO: Description of the local network, if any, connected to the WBCN node gateway machine.

Element: Centralized key distribution

Table: NODEPAIR

P/S: Scratch

Created: EXTRACT.COM

Table Description: This table is created during the key set extraction phase. It amounts to the cross product of the node table column with itself followed by deletion of the main diagonal. If one thinks of a table column as a column vector V, the NODEPAIR is the listing of the elements of the matrix

$$[V \times V] - [a_{ij}]_{i=j} \quad .$$

Observe that the result is a set of pairs of all possible communicating node pairs in the WBCN.

Column Definition: ORIG_NODE = C20
DEST_NODE = C20

Row Entries: Each row is an origination/destination node pair from the WBCN.

Entry Description: Because the participation in the WBCN on the part of a single node may be dynamic, some provision is needed to account for changes and expansion in the WBCN participation.

Element:	Centralized key distribution
Table:	INDEXTABLE
P/S:	Scratch
Created:	EXTRACT.EXE
Table Description:	This table provides a temporary holding structure for all the matched keys, nodes, and pointers prior to creating the separate VMS files.
Column Definition:	ORIG_NODE = C20 DEST_NODE = C29 JNDEX = I4 PERIOD = I4 KEY = C16
Row Entries:	Each row is a key and key pointer associated with a period and pair of WBCN nodes.
Entry Description:	The entries in this table consist of all the keys and pointers for the next key distribution cycle, paired with their respective nodes. The next step in the process is to extract from this table the pairwise distinct key sets.

Element:	Centralized key distribution
Table:	KEYDUMP
P/S:	Scratch
Table Description:	This is a temporary storage table for the distinct key sets. The distinct sets are extracted from INDEXTABLE and stored in KEYDUMP just prior to being written out to VMS files.
Column Definition:	The column definition is the same as those for INDEXTABLE.
Row Entries:	These are the same as for INDEXTABLE with the exception that all the row entries are for a single origination node.
Entry Description:	These are the same as for INDEXTABLE.

Element: Site-specific key management

Table: KEYTABLE

P/S: Permanent

Created: SETUP.COM

Table Description: This is the node-specific key table.

Column Definition

ORIG_NODE = C20
DEST_NODE = C20
KEYPTR = I4
PERIOD = I4
KEY = C16
COND = C3
NUSE = I4

Row Entries: Each row entry is a description of a key, its pointer, which node it can be used with, and use information.

Entry Description: ORIG_NODE: The node arbitrarily designated the origin node in the process of creating the NODEPAIR table.

DEST_NODE: The node arbitrarily designated the destination node in the process of creating the NODEPAIR table. Notice that either ORIG_NODE or DEST_NODE must be the same as the local node identifier in order for the key record to be included in the specific key distribution set.

KEYPTR: This is the key index assigned to the key by the INDEX.EXE program and hence carried over from the KEYINDEX table.

PERIOD: This is the number or designator of the period for the key.

KEY: The key in plain-text ASCII code.

COND: One of the two data items used to track the usage of the key.

NUSE: The other data item used to track the usage of the key.

APPENDIX B

LISTINGS OF ALL THE SOFTWARE THAT CONSTITUTES THE WCS

The key management DCL command processes used to establish and process the key distribution on the key management machine are

SETUP.COM
ROOT.COM
KEYLOAD.COM
UPDATE.COM
PERIOD.COM
EXTRACT.COM
DISTRIB.COM
ERROR.COM
LOGICALS.COM

```
$!!     ********************  SETUP.COM  ********************
$!!
$!!     This command file is used to set the basic underlying database
$!!     and file structure for the key management system.  This process
$!!     should be run to completion only once.  If it is run on the
$!!     active key management database all is lost.
$!!
$!!
$!!     There are only a few things that need to be done.  Primarily the
$!!     data base needs to be setup.
$!!
$!!
$!!     First setup the logical symbols for this process
$!!
$       @ud:[crypto.keymgr]logicals.com
$!!
$!!     Then visit with the user.
$!!
$       ws start
$       ws erase
$       ws bell
$       ws bell
$       ws "This process creates the database and constructs the permanent  "
$       ws "tables used by the key manager code to distribute keys  to the  "
$       ws "WBCN gateway nodes to use with the DES encryption.              "
$       ws "This process should be run only once.  If this process is run   "
$       ws "against an active key management environment,  all will be lost."
$       ws "I recommend that this process be removed form the system after  "
$       ws "it has been sucessfully run the first time.  There is an        "
$       ws "intercept inquiry next that gives the user a chance to quit here"
$       ws "if needed.  If you want to proceed enter 'READY',  if you want  "
$       ws "stop enter 'EXIT', any other response will loop on the inquiry. "
$       ws line
$       ws line
$!!
$       ques_loop:
$!!
$       inquire command "  Please enter command ( READY or EXIT )"
$!!
$       If command .eqs. "EXIT" then goto exit_point
$       If command .nes. "READY" then goto ques_loop
$!!
$!!     Otherwise continue
$!!
$       hush1
$       hush2
$       destroydb keymgr
$!!
$       hush1
$       hush2
$       createdb keymgr
$!!
$!!     Run the Ingres script that creates the tables -ptrtable-, and
$!!     -keyindex-. This script also initializes the values in -ptrtable-
$!!
$       hush1
$       hush2
```

```
$!!
$       ingres -s -d keymgr <setup.ing
$!!
$       ws line
$       ws line
$       ws bell
$       ws "The setup process has completed                                    "
$       exit_point:
$!!
$       exit
```

```
$!!     ******************** ROOT.COM ********************
$!!
$!!     This is the controlling command file the is used to run the
$!!     key management pieces of software.  This key management software
$!!     consists of three separate components.
$!!
$!!                                KEYLOAD.COMm
$!!                                EXTRACT.COMm
$!!                                UPDATE.COM
$!!                                DISTRIBUTE.COMm
$!!
$!!     KEYLOAD.COM is the command file that moves the key material from
$!!     the distribution tape into an INGRES table for storage and eventual
$!!     redistribution to the various WBCN gateway nodes
$!!
$!!     EXTRACT.COM is the command file that runs the various processes
$!!     that creates the mutually unique key sets that are destined for
$!!     the various WBCN gateway nodes
$!!
$!!     UPDATE.COM is the command file that provides a means for the key
$!!     management to update the WBCN gateway node information
$!!
$!!     PERIOD.COM is the command file that provides a means for the key
$!!     management to update the key period table
$!!
$!!     DISTRIBUTE.COM is the command file that runs the processes for
$!!     actually creating the node key distribution tapes
$!!
$!!
$!!     There are a selection of subordinate command files used to support
$!!     the processing in ROOT.COM.
$!!
$!!     LOGICALS.COM is used to hold all the common logical definitions
$!!     that are invoked in the main commmand files.
$!!
$!!     SETUP.COM contains the initial setup processing needed to establish
$!!     the key management function.  It should be run only once and then
$!!     removed from the system.  It will completely destroy the key
$!!     management tables if run twice.
$!!
$!!
$!!     Is is not intended that the user need to interrupt this process,
$!!     however control t and y have not been turned off.  The following
$!!     line will do that if it becomes necessary.
$!!
$!!!!!  set nocontrol=(t,y)
$!!
$!!
$!!     The first part of the command file sets up the logical definitions
$!!
$       @ud:[crypto.keymgr]logicals.com
$!!
$!!
$!!
$!!     All of the temporary files and tables used in this and the subordinate
$!!     command files and processed have the extension .tmp.  This makes it
$!!     easier to maintain the housekeeping as the processes run.
```

```
$!!
$!!
$!!     All of the Ingres script files have  .ing extensions.  These are
$!!     utilized throughout the processing.
$!!
$!!
$       ws start
$       ws erase
$       ws bell
$       ws bell
$!!
$       ws "This is the supervisory process for the WBCN key management"
$       ws "key distribution and management code."
$       ws line
$       ws line
$       ws "this code has four sections:                                "
$       ws line
        ws "keyload -- this reads the NSA key distribution tape and   "
        ws "           encorporates the contents into the key         "
        ws "           management tables                              "
        ws line
        ws "update  -- this section activates the database processes  "
        ws "           that provide for updating the WBCN gateway node"
        ws "           information                                    "
        ws line
$       ws "period  -- this section activates the database processes  "
$       ws "           that provide for updating the key period data  "
$       ws "           information                                    "
$       ws line
$       ws "extract -- this section extracts the pairwise unique sets "
$       ws "           of keys to be sent to the various WBCN nodes   "
$       ws "           and creates the VMS files that hold these keys "
$       ws line
$       ws "distribute -- this section actually writes the tapes to go"
$       ws "              to the specific WBCN gateway nodes           "
$       ws line
$       ws "exit    -- provides a controlled exit for the process     "
$       ws line
$       pause
$       ws line
$       ws "These processes are appropriately done in sequence in the "
$       ws "above described order.  However they do not need to be run"
$       ws "immediately after each other.  Two notes of caution: First"
$       ws "interrupting any of these processes with a control-c or a "
$       ws "control-y will more than likely have unpredictable and    "
$       ws "unpleasant results and is therefore is not recommended.   "
$       ws "Second, this process is dimensioned to handle up to a     "
$       ws "million keys on the NSA distribution tape.  Trying to load"
$       ws "more than that at one time will have very predictable and "
$       ws "unpleasant results.                                        "
$!!
$       ws line
$       ws line
$       ws "All the command responses used in the process must be in  "
$       ws "uppercase.  Please set the caps-lock on your keyboard.    "
$       pause
$       pause
```

```
$!!
$       get_command_loop:
$!!
$!!     Check to see if the error flag has been set.  If so branch to exit
$!!     before clearing the screen
$!!
$       if error_cond .eqs. "ON"  then goto exit_loop
$!!
$!!     Otherwise continue processing
$!!
$       ws start
$       ws erase
$!!
$!!
$       ws "Enter the command for the section of this key management  "
$       ws "process that you desire to use:                           "
$       ws line
$       ws "     KEYLOAD -- process the NSA distribution tape         "
$       ws line
$       ws "     UPDATE  -- update the gateway node information       "
$       ws line
$       ws "     PERIOD  -- update the key period information
$       ws line
$       ws "     EXTRACT -- prepare a new set of keys for distribution"
$       ws line
$       ws "     DISTRIB -- write the tapes for distrbution to the    "
$       ws "                WBCN nodes                                "
$       ws line
$       ws "     EXIT    __ provides a controlled exit                "
$       ws line
$       ws line
$!!
$!!
$!!
$       inquire command -
        "Enter command ( KEYLOAD, UPDATE, PERIOD, EXTRACT, DISTRIB, EXIT)"
$!!
$       if command .eqs. "KEYLOAD" then goto keyload_loop
$!!
$       if command .eqs. "EXTRACT" then goto extract_loop
$!!
$       if command .eqs. "UPDATE"  then goto update_loop
$!!
$       if command .eqs. "PERIOD"  then goto period_loop
$!!
$       if command .eqs. "DISTRIB" then goto distrib_loop
$!!
$       if command .eqs. "EXIT"    then goto exit_loop
$!!
$!!     Otherwise the entered is a bad command.  So notify the user and loop
$!!     through the get_command_loop again
$!!
$!!
$       ws bell
$       ws line
$       ws "The command you entered was not one of the choices, please "
$       ws "try again                                                  "
```

```
$       pause
$!!
$       goto get_command_loop
$!!
$!!
$       keyload_loop:
$!!
$!!     This loop calls the keyload command file.  Upon completion it
$!!     recycles through the get_command_loop.  All of the progess information
$!!     is in the called  .com process
$!!
$       @keyload.com
$       goto get_command_loop
$!!
$!!
$       extract_loop:
$!!
$!!     This loop calls the extract command file.  Upon completion it
$!!     recycles through the get_command_loop.  All of the progress information
$!!     is in the calledg  .com process.
$!!
$       @extract.com
$       goto get_command_loop
$!!
$!!
$       update_loop:
$!!
$!!     This loop calls the update command file.  Upon completion it
$!!     recycles through the get_command_loop.  All of the progress information
$!!     is in the called  .com process.
$!!
$       @update.com
$       goto get_command_loop
$!!
$       period_loop:
$!!
$!!     This loop calls the period command file.  Upon completion it
$!!     recycles through the get_command_loop.  All of the progress information
$!!     is in the called  .com process.
$!!
$       @period.com
$       goto get_command_loop
$!!
$!!
$       distrib_loop:
$!!
$!!     This loop calls the distribution command file.  Upon completion it
$!!     recycles through the  get_command_loop.  All of the progress information
$!!     is in the called  .com process.
$!!
$       @distrib.com
$       goto get_command_loop
$!!
$!!
$       exit_loop:
$!!
$!!     This loop provides for and orderly exit from the root.com process
```

```
$!!
$!!
$       exit
```

```
$!!     ********************  KEYLOAD.COM  ********************
$!!
$!!     This command file runs the processes that unload the NSA
$!!     distribution tape and installs the keys in the appropriate
$!!     database tables

$!!
$!!
$       ws start
$       ws erase
$!!
$       ws "This process unloads the NSA key distribution tape and moves "
$       ws "the keys into database tables for storage and further        "
$       ws "distribution to the WBCN nodes                               "
$       ws line
$       ws line
$!!
$!!
$!!     The first step in the process is to move the keys from the tape to
$!!     a VMS file. The tape is unloaded into the VMS file -keyfile.tmp-
$!!     But before that I need to  make sure that no old temporary files
$!!     are lingering around
$!!
$!!
$       hush1
$       hush2
$       delete 'path'keyfile.tmp;*
$!!
$!!
$!!     Now start to setup for the tape handling.  This loop gives the
$!!     user the opportunity to get the tape mounted and ready to copy
$!!
$       get_tape:
$!!
$!!     Set the default for the tape drive
$!!
$       tape_in = "mta0:"
$!!
$!!     Poll the user for the actual tape drive identification
$!!
$       ws bell
$!!
$       ws "This process needs the identifer of the tape drive you intend"
$       ws "to use.  This identifer should look like ( mta0:, mfa0:,     "
$       ws "mta1: or such).  This process defaults to mta0:. If that is  "
$       ws "the correct identifier then a CR is all that is needed for   "
$       ws "the response to the following question                       "
$       ws line
$       ws line
$       inquire drive "Please enter the correct identifer"
$!!
$!!
$       if drive .nes. "" then tape_in = drive
$!!
$!!
$!!
$!!     May want to put some error checking in here
```

```
$!!
$!!
$       ws line
$       ws line
$       ws "The process is ready to copy the tape.  Please load the tape"
$       ws "on the drive; be sure that the drive is online and ready    "
$       ws "to proceed                                                  "
$       ws line
$       ws "This process has a loop built in here.  If you do not enter "
$       ws "-READY- in response to the following inquiry the process    "
$       ws "will loop to ask you for the tape identifier again.  If you "
$       ws "need additional time to do something this is a convenient   "
$       ws "place to suspend this process.                              "
$       ws line
$       ws line
$       inquire tape_load   "When you are ready enter ( READY) "
$!!
$       if tape_load .nes. "READY"  then goto get_tape
$!!
$!!
$!!     Qtherwise the tape is mounted on the correct drive and is ready to
$!!     go.
$!!
$       ws line
$       ws line
$       ws "the copy has started"
$       pause
$!!
$       hush1
$       hush2
$       mount/for/den=1600 'tape_in'
$!!
$       hush1
$       hush2
$       copy 'tape_in'   'path'keyfile.tmp
$!!
$!!     The copy is complete so dismount the tape
$       dismount 'tape_in'
$!!
$       ws line
$       ws line
$       ws bell
$       ws "The copy is complete.  Take the tape off of the drive"
$       ws line
$       ws  "The process will now index the keys and store them  "
$       ws  "in the Ingres table -keyindex-.  This may take      "
$       ws  "awhile to do , be patient                           "
$       ws line
$       pause
$!!
$!!
$!!
$!!     Next this process indexes the keys in the data structure
$!!     and stores the keys in their resident table -keyindex-.
$!!
$!!     First get rid of any temporary files
$!!
```

```
$       hush1
$       hush2
$       delete 'path'keyindex.tmp;*
$!!
$!!     Run the indexing program
$!!
$       hush1
$       hush2
$       run/nodebug index
$!!
$!!     And clean up any temporary files on the way out.
$!!
$       hush1
$       hush2
$       delete 'path'keyindex.tmp;*
$       hush1
$       hush2
$       delete 'path'keyfile.tmp;*
$!!
$!!     The key loading has been completed.  Control is transfered
$!!     back to the calling program
$!!
$!!     Signal to the user and return
$       ws bell
$       ws line
$       ws line
$       ws "The keyload process has finished"
$!!
$       exit
```

```
$!!     ********************  UPDATE.COM  ********************
$!!
$!!     This command file inables the user to run Ingres on the
$!!     node database to make any updates and corrections to the
$!!     WBCN node data  before generating the key distribution tapes.
$!!
$!!
$       assign/user sys$command sys$input
$       qbf -s keymgr -f qbfnodemgr
$       exit
```

```
$!!     ********************  PERIOD.COM  ********************
$!!
$!!     This command file inables the user to run ingres on the
$!!     key period table.  The user must be sure that there are
$!!     enough specified key periods to cover the range of the
$!!     current key assignment process.  Since it is fairly easy
$!!     to add more key period information I recommend that the
$!!     user provides for many more that the necessary key periods.
$!!
$       assign/user sys$command sys$input
$       qbf -s keymgr prdtable
$       exit
```

```
$!!     ********************  EXTRACT.COM ********************
$!!
$!!     This command file runs the processes that determine the key indexes
$!!     on the sets of WBCN nodes that match with the indexes of the keys
$!!     to be distributed with this pass of the key management processes.
$!!     This process then extracts the correct number of keys from the
$!!     primary key table -keyindex- and prepares the pairwise unique key sets
$!!     for distribution to the participating nodes.
$!!
$!!     The bulk of the work is done in the program -extract-.  The program
$!!     examines the key distribution requirements and appends to the
$!!     node-pair table an index that corresponds to the key that will be
$!!     assigned to the node pair.  Then the program  pairs the indexed node
$!!     pairs and the indexed keys to produce the pairwise unique VMS files
$!!     for distribution to the WBCN nodes.
$!!
$       ws start
$       ws erase
$       ws line
$       ws line
$       ws "This process creates pairwise unique keysets for the wbcn nodes."
$       ws "You should have verified , with the UPDATE process that all the "
$       ws "WBCN node information is current and complete.  This process    "
$       ws "allows for a continue or exit choice just in the case you need  "
$       ws "to confirm the node information.  If you want to continue enter "
$       WS "READY, any other response will end this process.                "
$!!
$       ws line
$       inquire command  "Enter READY if you want to continue"
$!!
$       if command .nes. "READY" then goto  exit_point
$!!
$!!     Otherwise continue with the extraction process.
$!!
$!!
$!!     First build a temporary table of the nodepairs with a call to ingres
$!!
$       hush1
$       hush2
$!!
$       ingres -s -d keymgr  <nodepair.ing
$!!
$!!
$!!     Before starting the extract process delete any residual key files.
$!!     This is done out here but not in or after extract.  This way the
$!!     VMS key distribution files are retained until a new set is written.
$!!     Also delete the .SAV files that are the backup residual files from
$!!     the previous key distribution.  Both of these are not confirmed deletes.
$!!
$!!
$       hush1
$       hush2
$       delete 'path'*.key;*
$!!
$       hush1
$       hush2
$       delete 'path'*.prd;*
```

```
$!!
$       hush1
$       hush2
$       delete 'path'key*.sav;*
$!!
$       hush1
$       hush2
$       delete 'path'prd*.sav;*

$!!     Now run the program that creates the matching indexes of the nodepairs
$!!
$!!
$       ws line
$       ws line
$       ws "The process that creates and writes the VMS files starts now."
$       ws "It may take a while, be patient.                              "
$!!
$       hush1
$       hush2
$       run/nodebug extract
$!!
$!!     Get rid of the extract scratch files
$!!
$       hush1
$       hush2
$       delete 'path'nodeindex.tmp;*
$!!
$!!
$!!
$!!     There is an error-flag that can be set by pairindex.  This flag is set
$!!     if there are not enough keys available for distribution as required
$!!     The flag is the process logical -error_cond-. The flag is initialized
$!!     in LOGICALS.COM that is run first by ROOT.COM.  If the flag is set to
$!!     "ON" this process returns an error message and exits.
$!!
$       if error_cond .eqs. "ON" then goto exit_point
$!!
$!!
$!!     This process has completed.  The VMS files are ready to move to tape.
$!!     This move is accomplished by running the distribute command file.
$!!
$       ws line
$       ws line
$       ws "This process has completed.  The VMS files are ready to move to"
$       ws "tape using the DISTRIB option.                                 "
$!!
$       exit_point:
$!!
$       exit
```

```
$!!     ********************  DISTRIB.COM  ********************
$!!
$!!     This process is invoked to actually write the prepared VMS
$!!     files onto tape for distribution to the WBCN gateway nodes.
$!!
$!!     The keys, pointers and all the other identification information
$!!     for each node is contained in a VMS file that is
$!!     identified by ud:[crypto.keymgr]NODEMNAME.key. There should only
$!!     be one version or issue of any of these key files since any
$!!     lingering versions are deleted during the process that writes the
$!!     newest versions.  This process pesters the user a bit about
$!!     getting all the needed tapes and other materials together before
$!!     actually starting the process.  It may be appropriate to remove
$!!     some of this dialogue eventually.
$!!
$!!
$!!
$       ws start
$       ws erase
$       ws bell
$       ws bell
$!!
$       ws "You have started the process that writes the node-specific"
$       ws "key distribution tapes for the many WBCN gateway nodes.   "
$       ws line
$       ws line
$       ws "First the process determines how many tapes you need to   "
$       ws "have on hand before you actually start the file movements."
$       ws "line                                                      "
$       ws "In addition to the tapes you need to have the blank labels"
$       ws "to put on the tapes as they are written.  It is fairly    "
$       ws "important to not mix up these tapes so I recommend that   "
$       ws "they be labeled as they are written.  Don't forget you    "
$       ws "also need something to write with and you need a          "
$       ws "write-ring for the tapes.                                 "
$!!
$       pause
$       pause
$!!
$!!     Now find out how many tapes are needed for this pass
$!!
$!!
$       count = 0
$       first_loop:
$       firstfile = f$search("*.key")
$       if firstfile .eqs. "" then goto next_one
$       count = count + 1
$       if count .ge. 50  then goto next_one
$       goto first_loop
$       next_one:
$!!
$!!
$!!     If count remains zero then something is very wrong.  Likely
$!!     there are no VMS key files and the whole process is suspect.
$!!     Issue an error message and get out.
$!!
$       if count .nes. 0  then goto next_two
```

```
$!!
$!!     Otherwise there is an error.  Set error_cond to on, issue
$!!     a message and exit
$!!
$       ws erase
$       ws start
$       ws bell
$       ws bell
$       ws bell
$       ws "There is a processing error.  There are no VMS key files "
$       ws "ready for distribution.  This process now creates a      "
$       ws "gentle abort                                             "
$!!
$       pause
$!!
$       error_cond == "ON"
$!!
$       goto exit_point
$!!
$!!
$       next_two:
$!!
$!!     To get here there is at least one VMS key file ready for writing
$!!     to tape.
$!!
$!!
$!!
        prep_loop:
$!!
$!!
$       ws line
$       ws line
$       ws line
$       ws "You will need  ",count," tapes for this process when you "
$       ws "have the tapes, labels, a write-ring, and a marker ready "
$       ws "to go enter READY.  If you enter EXIT this process will  "
$       ws "gracefully exit without writing the tapes or altering    "
$       ws "anything else.  If you enter enything but  READY or EXIT "
$       ws "this process will loop on this input.                    "
$!!
$       pause
$       ws line
$       ws line
$!!
$       inquire command "Please enter command (READY or EXIT)"
$!!
$       If command .eqs. "EXIT" then goto exit_loop
$       if command .nes. "READY" then goto prep_loop
$!!
$!!
$!!     To get here there are VMS key files to write to tape and the
$!!     user thinks he has the tapes and materials ready.
$!!
$!!
$!!     Now set up the tape write loop
$!!
$!!
```

```
$       out_tape == "MTA0:"
$       dens == "1600"
$!!
$!!     This process needs the identifier for the tape drive used to
$!!     write these tapes .  I've set it up to default to MTA0:.
$!!     I then poll the user for the correct drive identifer.  This default
$!!     could be set to whatever the key distribution machine is setfor
$!!     then this section would not be necessary.
$!!
$       ws "This process needs the identifier of the tape drive you intend"
$       ws "to use.  This identifer should look like  ( mta0:, mfa0:,     "
$       ws "mta1: or such )                                               "
$       inquire drive "Please enter the correct tape drive identifer"
$!!
$       if drive .nes. "" then  out_tape = drive
$!!
$       write_loop:
$!!
$!!     Extract the node file name
$!!
$       dum = f$search("*.KEY") - f$directory()
$       file = f$extract(f$locate(":",DUM) + 1,f$length(DUM),DUM)
$!!
$!!     Test  the contents of -file- if it is blank there are
$!!     no more key files left to write so quit
$!!
$       if file .eqs. "" then goto closeout
$!!
$!!     Next extract the nodename
$!!
$       node_name = f$extract(0,f$locate(".",file),file)
$!!
$!!
$       ws line
$       ws line
$       ws "The current key file is for   ",node_name
$!!
$       pause
$!!
$       mount_loop:
$!!
$       ws line
$       ws line
$       inquire command "Mount the tape and enter READY when you are ready"
$!!
$       if command .nes. "READY" then goto mount_loop
$!!
$!!
$!!     To get here the files are there, the tape is mounted and the
$!!     node has been identified to the user
$!!
$!!     The actual tape write is here
$!!
$       initialize 'out_tape'  'node_name' /density='dens'
$       mount 'out_tape' 'node_name'
$       copy 'node_name'.key  'out_tape'/log
$       copy 'node_name'.prd  'out_tape'/log
```

```
$       dismount/nounl  'out_tape'
$!!
$!!     This finishes the tape write for that node.  Now copy the new key
$!!     to an old file and delete the current .KEY file.  This saves the
$!!     old key sets if they are needed and still lets the DCL process use
$!!     the .KEY extensions to keep things sorted out.
$!!
$       copy 'node_name'.key  key'node_name'.sav
$       copy 'node_name'.prd  prd'node_name'.sav
$!!
$!!     The delete is a confirm mode so if you have a problem you still
$!!     have a fallback option.
$!!
$       ws line
$       ws line
$       ws "You will now be asked if you want to delete the residual key  "
$       ws "files.  If the tape write has been done sucessfully then you  "
$       ws "should delete these files.                                    "
$       ws line
$       ws line
$       delete/confirm 'node_name'.key;*
$       ws line
$       ws line
$       delete/confirm 'node_name'.prd;*
$!!
$!!
$!!
$       cont_loop:
$       ws bell
$       ws bell
$       ws line
$       ws line
$       ws "The tape for   ",node_name," has been written.  Take the"
$       ws "tape off of the tape drive and label it before you lose  "
$       ws "track of which tape was just written.  Do this before you"
$       ws "start another tape write.                                "
$!!
$       pause
$!!
$       inquire command  "Enter READY when you are ready to proceed      "
$!!
$       if command .eqs "READY"  then goto write_loop
$!!
$!!     Otherwise loop on this instruction
$!!
$       goto cont_loop
$!!
$       closeout:
$       ws line
$       ws line
$       ws "There are no remaining key files.  The process in finished"
$       pause

$       exit_point:
$       exit
```

```
$!!     ******************** ERROR.COM ********************
$!!
$!!     This command file is called by the process -pairindex-.  All
$!!     it does is notify the user of the shortage of available keys.
$!!
$       ws erase
$       ws start
$       ws bell
$       ws bell
$       ws "THERE ARE NOT ENOUGH KEYS AVAILABLE FOR THE DISTRIBUTION"
$       ws line
$       ws line
$       ws "The process will create a graceful abort at this point  "
$       ws line
$       ws "Adjust the key allocation parameter or load additional  "
$       ws "keys before restarting this procedure                   "
$!!
        error_cond == "ON"
$!!
$       pause
$       pause
$       pause
$!!
$       exit
```

```
$!!     ********************  LOGICALS.COM  ********************
$!!
$!!     This .COM file contains all the process logical definitions
$!!     that are used by the key management code.
$!!
$       PATH == "UD:[CRYPTO.KEYMGR]"
$       WS == "WRITE SYS$OUTPUT"
$       BLANK_LINE == ""
$       BELL == ""
$       ERASE == ""
$       START == ""
$       LINE == ""
$       hush1 == "assign/user nl:  sys$error"
$       hush2 == "assign/user nl:  sys$output"
$       pause == "wait 00:00:05"
$       error_cond == "OFF"
```

The programs used by the DCL key management processes are

EXTRACT.QF
INDEX.QF

```
c       ********************   extract.qf   ********************
c
c       This program performs a selection of services:  It determines
c       the number of keys that can be distributed during this cycle,
c       It determines the range of indexes that are to be used this
c       cycle, It appends an index to the nodepairs entries in the
c       -nodepair- table.  This set of indexed pairs is stored in a
c       temporary table  -nodeindex- :  it then maps the nodepairs to
c       the keys with this assigned index:  finally it writes out the
c       node specific VMS key files.
c
c       This process creates a temporary file nodeindex.tmp as a working
c       buffer during excution.
c
c       This process creates the node specific VMS key files.  These files
c       have the form
c
        program extract
c
c       setup and declarations
c
c
##      declare
c
##      character*33    outstring1, outstring2
c
##      character*1     blank
c
##      character*20    fileid1
c
##      character*9     fileid2, fileid3
c
##      character*10    fhighptr, flowptr, fnbrkeys, fnextprd, fnbrprds
c
##      character*16    lockey(250000)
c
##      character*20    forig(250000), fdest(250000), fnode(1000)
c
##      character*25    max_date
c
##      integer*4       highidx, lowidx, nnbrkeys, nxtprd, nperiods
##      integer*4       lowindex, nextperiod, max_period
c
##      integer         availkeys, nnodes, npairs, locidx, needkeys
##      integer         key_index, period_index, strlen, need_period
c
##      integer         m, i, j, k
c
        data    fhighptr, flowptr, fnextprd, fnbrkeys, fnbrprds
     &       / 'highptr','lowptr','nextprd','nbrkeys','nbrprds'/
c
        data            fileid1 / 'dua1:[crypto.keymgr]'  /
        data            fileid2 / '.key,text'             /
        data            fileid3 / '.prd,text'             /
        data            blank   / ' '                     /
c
c
```

```
##      ingres keymgr
c
c       Define the range variables for the Ingres references
c
##      range of pt is ptrtable
##      range of nt is nodetable
##      range of np is nodepair
##      range of ki is keyindex
##      range of pd is prdtable
c
c
c       Get some of the pointer and count information
c
##      retrieve ( highidx = pt.ptr ) where pt.attr = fhighptr
##      retrieve (  lowidx = pt.ptr ) where pt.attr = flowptr
##      retrieve (  nxtprd = pt.ptr ) where pt.attr = fnextprd
##      retrieve (nnbrkeys = pt.ptr ) where pt.attr = fnbrkeys
##      retrieve (nperiods = pt.ptr ) where pt.attr = fnbrprds
c
c
c       At this point the following has been retrieved into local
c       variables
c
c               highidx == This is the lowest unassigned index value for
c                          the key table.  Since the keys are indexed
c                          sequentially, this is the next assignable
c                          index for the key list.
c
c               lowidx == This the lowest valued unassigned index available
c                          in the key table.  Any keys with lower index
c                          have been deleted from the table.  The
c                          difference between highidx and lowidx is
c                          the number of keys available for distribution.
c
c               nxtprd  == This carries the number of the next assignable
c                          key period
c
c              nnbrkeys == This carries the maximum number of keys that
c                          can be distributed per pair of nodes per period.
c
c              nperiods == This carries the number of key periods assigned
c                          during this key distribution pass
c
c
c       Calculate the number of keys available for the distribution
c
        availkeys = highidx - lowidx
c
c       Determine the number of keys needed for this distribution
c       and also move the node identifiers into local storage
c
        i = 1
c
##      retrieve ( fnode(i) = nt.node_name )
##      {
        i = i + 1
##      }
```

```
c
c       Adjust for the loop process
c
        nnodes = i - 1
c
c       Then the total number of needed keys is given by
c
        needkeys = nnodes * (nnodes - 1) * nperiods * nnbrkeys

c
c
c       Test the key numbers to see if there are enough keys available
c       for distribution.  If not issue error messages and exit.
c       Otherwise continue processing
c
        if (needkeys . gt. availkeys ) then
c
                call lib$spawn ('$@error.com')
                go to 100
c
        end if
c
c       To get here there are enough keys in the indexed table.  So
c       move all the nodepairs into temporary local storage and get the
c       exact count.

c
        i = 1
##      retrieve (forig(i) = np.orig_node,fdest(i)=np.dest_node)
##      {
        i = i + 1
##      }
c
c       Ajust for the last pass of the loop
c
        npairs = i - 1
c
c
c       Now move the needed keys into local storage
c
        i = 1
c
##      retrieve ( lockey(i) = ki.key) where ki.ptr >= lowidx
##                              and ki.ptr <= lowidx + needkeys
##      {
        i = i + 1
##      }
c
c


c       Now setup the loop that runs through the generation of the
c       index pairing and the key assignment
c
c       The first assignable key has the index lowidx. First loop
c       through the count of the node pairs.  The next loop advances
c       over the count of the periods for key distribution.  The
c       inner-most loop advances over the count of how many keys
```

```
c       belong to each period.
c
c
c       Open the VMS file that recieves the output of the indexing loop
c
        open(unit=50,file='nodeindex.tmp',status='new',carriagecontrol='none',
     &       recordtype='fixed',recl=76     )
c
c       Put in the format for the write
c
50      format(a,a,i10,i10,a)
c
        locidx = 1
c
        do  j = 1, npairs
c
c
                do  k = 1, nperiods
c
                period_index = nxtprd + ( k - 1 )
c
                        do  m = 1, nnbrkeys
c
                        key_index = lowidx + (locidx - 1)
c
        write(50,50) forig(j),fdest(j),key_index,period_index,
     &               lockey(locidx)
c
                        locidx = locidx + 1
c
                        end do
c
                end do
c
        end do
c
c
c
c       At the completion of these loops all of the key indexes have been
c       assigned
c
c       Close the VMS file
c
        close(unit=50)
c
c       Now create the data table used for storing these indexed pairs
c       and copy the VMS file into it.
c
c
##      destroy indextable
##      create indextable(orig_node=c20,dest_node=c20,jndex=i4,period=i4,
##                        key=c16)
c
##      copy indextable(orig_node=c20,dest_node=c20,jndex=c10,period=c10,
##                      key=c16)
##           from "dua1:[crypto.keymgr]nodeindex.tmp,text"
c
```

```
c
c       At this point all of the indexed node information is stored
c       in the Ingres table -indextable-.  Now make sure that the period
c       table is up to date and if not bring it up to date with the
c       default -one month- value.  Then cleanup and build the
c       output files.
c
c
##      retrieve (max_period = max(pd.period))
##      retrieve (max_date  =  pd.period_date )
##              where pd.period = max_period
c
c       Calculate the needed period information then compare it with
c       the information in the table.  If the table is short, add in
c       the needed additional values.
c
        need_period = nxtprd + nperiods - 1
c
        if  ( need_period .ge. max_period )  then
c
                do j = max_period + 1, need_period

##              append to prdtable (period = j,
##                      period_date = date(max_date) +
##                      concat(ascii(j - max_period)," month") )
c
                end do
        end if
c
c
##      modify nodepair to truncated
##      destroy nodepair
c
c
##      delete ki  where ki.ptr >= lowidx
##                              and ki.ptr <= lowidx + needkeys
c
c
c
c       This leaves only the VMS file nodeindex.tmp to delete which
c       will be done in EXTRACT.COM.
c
c       The last part of this process is writing the VMS files for
c       each of the WBCN nodes
c
c
c
c       The way this works as follows.  All the information for a single
c       node that is stored in the table -indextable- is retrieved into
c       a tempory dummy table.  Then the contents of the table is written
c       out to a VMS file.  Then the dummy table is emptied and the loop
c       done again for the next node.
c
c
##      range of it is indextable

##      destroy keydump
```

```
c
##      create keydump (orig_node=c20,dest_node=c20,keyptr=i4,period=i4,
##                      key=c16)
c
c
c
        do  j = 1, nnodes
c
##      modify keydump to truncated
c
##      append to keydump(orig_node=it.orig_node,dest_node=it.dest_node,
##                        keyptr=it.jndex,period=it.period,key=it.key)
##              where it.orig_node=fnode(j)
##              or    it.dest_node=fnode(j)
c
c
c       All of the location, key-indexes and period-pointers for this one
c       node are in the keydump table.  Now write out to the VMS file.
c
c
c       find out the length of the node identifier
c
        strlen = index(fnode(j),blank) - 1
c
c       Build up the output file identifier
c
        outstring1 = fileid1//fnode(j)(1:strlen)//fileid2
        outstring2 = fileid1//fnode(j)(1:strlen)//fileid3
c
##      copy keydump(orig_node=c0,sp=d1,dest_node=c0,sp=d1,
##                   keyptr=c0,sp=d1,period=c0,sp=d1,key=c0,nl=d1)
##         into outstring1
c
##      copy prdtable(period=c0,sp=d1,period_date=c0,nl=d1)
##         into outstring2
c
c       Then do it all over again for the next node
c
        end do
c
c
c
c       Finish up with the final bit of housekeeping
c
c       Adjust the pointers to reflect the actions of this process
c
        lowindex = lowidx + needkeys
        nextperiod = nxtprd + nperiods
c
##      replace pt( ptr = lowindex ) where pt.attr = flowptr
##      replace pt( ptr = nextperiod ) where pt.attr = fnextprd
c
##      modify keydump to truncated
##      destroy keydump
##      modify indextable to truncated
##      destroy indextable
c
```

```
c       This is the transfer location for error-exits
c
100     continue
        stop
        end
```

```
c       ********************  index.qf  ********************
c
c       this program appends an index to the key table.  It reads the
c       keys form the temporary storage table -keytemp- that was created
c       earlier in the KEYLOAD.COM process.  This program creates the
c       permament key table named -keyindex-.  The program indexes the
c       keys with a consecutive index.  The current maximum value of this
c       index is stored in the one-item table -keycount-.  This maximum
c       value is used as the starting value for the indexing value. Hence
c       the key index hopefully will not need to be repeated.
c
c       This program halls all of the keys from the temporary table -keytemp-
c       into local storage [in fkey]; fetches the current maximum index
c       value from the table -keycount-; then runs through a loop that
c       associates the index with the key.  This associated pair is
c       written out to a temporary VMS file and then stuffef into the
c       final table keyindex.  This last part is a lot faster than appending
c       each key/index pair to the table separately.
c
c
c
        program index
c
c       Set up the various parts to use ingres
c
##      declare
c
c       The declared fortran variables used in the program are:
c
c               kindex == the incremented index that is associated with
c                         each key.  I used kindex because index is an
c                         ingres reserved word
c
c               fkey   == the buffer used to hold all of the keys from the
c                         temporary table
c
c              highidx == the local value for the current stored maximum
c                         key index
c
c              lowpidx == the local value for the current stored minimum
c                         key index
c
c              fhighptr == the character string used to indicate the high
c                         value attr
c
c              flowptr == the character string used to indicate the low
c                         value attr
c
c              endidx  == the upper extent of the indexing loop
c
c
c
##      integer         iost
##      integer*4       kindex, highidx, lowidx,  endidx
##      character*16    fkey
##      character*10    fhighptr, flowptr
c
```

```
c
      data fhighptr, flowptr / 'highptr', 'lowptr' /
c
c
##    ingres  keymgr

##    range of pt is ptrtable
c
c
c     Now get the current maximum index value
c
##    retrieve ( highidx = pt.ptr ) where pt.attr = fhighptr
c
c
c     Open the file for input to read the keys.
c
      open (unit=70,file='keyfile.tmp',status='old')
c
c
c     Open a file for output
c
      open( unit=60,file='keyindex.tmp',status='new',
     &      carriagecontrol='none',recordtype='fixed',recl=26)

c     set format for the read
c
20    format(a)
c
c     Set format for the write
c
10    format(i10,a)
c
c
c     Initialize the counter and status flags for the loop
c
      iost = 0
c
      kindex = highidx
c

            do while  (iost .eq. 0 )
c
            read (70,20,iostat=iost) fkey

            write (60,10) kindex, fkey
c
            kindex = kindex + 1
c
            end do
c
30    continue

c     now close the files
      close(unit=70)
      close(unit=60)
c
c     Save the last value of the index
```

```
c
        endidx = kindex

c       Having built the file, copy it into the table
c
##      copy keyindex(ptr=c10,key=c16) from
##                      "dua1:[crypto.keymgr]keyindex.tmp,text"
c
c       Now do the housekeeping and exit
c
c       Store the new maximum key pointer value in the keycount table
c
##      replace pt(ptr = endidx ) where pt.attr = fhighptr
c
c
c       Get rid of the temporary key holding table
c
c
c       This process leaves all the newly indexed keys in the permanent
c       table -keyindex-.  The only file that needs to be cleaned up is
c       keyindex.tmp that was used to move the keys around
c
c

        stop
        end
```

The key user DCL command processes used to establish and process keys on the key utilization WBCN gateway nodes are

SETUP.COM
KEYLOAD.COM
LOGICALS.COM

```
$!!     ******************** SETUP.COM ********************
$!!
$!!     This command file is used to set the basic underlying database
$!!     and file structure for the key management system.  This process
$!!     should be run to completion only once.  If it is run on the
$!!     active key management database all is lost.
$!!
$!!
$!!     There are only a few things that need to be done.  Primarily the
$!!     data base needs to be setup.
$!!
$!!
$!!     First setup the logical symbols for this process
$!!
$       @ud:[crypto.keymgr]logicals.com
$!!
$!!     Then visit with the user.
$!!
$       ws start
$       ws erase
$       ws bell
$       ws bell
$       ws "This process creates the database and constructs the permanent  "
$       ws "tables used by the key manager code to distribute keys  to the  "
$       ws "WBCN gateway nodes to use with the DES encryption.              "
$       ws "This process should be run only once.  If this process is run   "
$       ws "against an active key management environment,  all will be lost."
$       ws "I recommend that this process be removed form the system after  "
$       ws "it has been sucessfully run the first time.  There is an        "
$       ws "intercept inquiry next that gives the user a chance to quit here"
$       ws "if needed.  If you want to proceed enter 'READY',  if you want  "
$       ws "stop enter 'EXIT', any other response will loop on the inquiry. "
$       ws line
$       ws line
$!!
$       ques_loop:
$!!
$       inquire command "  Please enter command ( READY or EXIT )"
$!!
$       If command .eqs. "EXIT" then goto exit_point
$       If command .nes. "READY" then goto ques_loop
$!!
$!!     Otherwise continue
$!!
$       hush1
$       hush2
$       destroydb keymgr
$!!
$       hush1
$       hush2
$       createdb keymgr
$!!
$!!     Run the Ingres script that creates the tables -ptrtable-, and
$!!     -keyindex-. This script also initializes the values in -ptrtable-
$!!
$       hush1
$       hush2
```

```
$!!
$       ingres -s -d keymgr <setup.ing
$!!
$       ws line
$       ws line
$       ws bell
$       ws "The setup process has completed                                    "
$       exit_point:
$!!
$       exit
```

```
$!!     ********************  KEYLOAD.COM  ********************
$!!
$!!     This command file runs the processes that unload the NSA
$!!     distribution tape and installs the keys in the appropriate
$!!     database tables

$!!
$!!
$       ws start
$       ws erase
$!!
$       ws "This process unloads the NSA key distribution tape and moves "
$       ws "the keys into database tables for storage and further        "
$       ws "distribution to the WBCN nodes                               "
$       ws line
$       ws line
$!!
$!!
$!!     The first step in the process is to move the keys from the tape to
$!!     a VMS file. The tape is unloaded into the VMS file -keyfile.tmp-
$!!     But before that I need to  make sure that no old temporary files
$!!     are lingering around
$!!
$!!
$       hush1
$       hush2
$       delete 'path'keyfile.tmp;*
$!!
$!!
$!!     Now start to setup for the tape handling.  This loop gives the
$!!     user the opportunity to get the tape mounted and ready to copy
$!!
$       get_tape:
$!!
$!!     Set the default for the tape drive
$!!
$       tape_in = "mta0:"
$!!
$!!     Poll the user for the actual tape drive identification
$!!
$       ws bell
$!!
$       ws "This process needs the identifer of the tape drive you intend"
$       ws "to use.  This identifer should look like ( mta0:, mfa0:,     "
$       ws "mta1: or such).  This process defaults to mta0:. If that is  "
$       ws "the correct identifier then a CR is all that is needed for   "
$       ws "the response to the following question                       "
$       ws line
$       ws line
$       inquire drive "Please enter the correct identifer"
$!!
$!!
$       if drive .nes. "" then tape_in = drive
$!!
$!!
$!!
$!!     May want to put some error checking in here
```

```
$!!
$!!
$       ws line
$       ws line
$       ws "The process is ready to copy the tape.  Please load the tape"
$       ws "on the drive; be sure that the drive is online and ready    "
$       ws "to proceed                                                  "
$       ws line
$       ws "This process has a loop built in here.  If you do not enter "
$       ws "-READY- in response to the following inquiry the process    "
$       ws "will loop to ask you for the tape identifier again.  If you "
$       ws "need additional time to do something this is a convenient   "
$       ws "place to suspend this process.                              "
$       ws line
$       ws line
$       inquire tape_load   "When you are ready enter ( READY) "
$!!
$       if tape_load .nes. "READY"  then goto get_tape
$!!
$!!
$!!     Qtherwise the tape is mounted on the correct drive and is ready to
$!!     go.
$!!
$       ws line
$       ws line
$       ws "the copy has started"
$       pause
$!!
$       hush1
$       hush2
$       mount/for/den=1600 'tape_in'
$!!
$       hush1
$       hush2
$       copy 'tape_in'   'path'keyfile.tmp
$!!
$!!     The copy is complete so dismount the tape
$       dismount 'tape_in'
$!!
$       ws line
$       ws line
$       ws bell
$       ws "The copy is complete.  Take the tape off of the drive"
$       ws line
$       ws  "The process will now index the keys and store them  "
$       ws  "in the Ingres table -keyindex-.  This may take      "
$       ws  "awhile to do , be patient                           "
$       ws line
$       pause
$!!
$!!
$!!
$!!     Next this process indexes the keys in the data structure
$!!     and stores the keys in their resident table -keyindex-.
$!!
$!!     First get rid of any temporary files
$!!
```

```
$       hush1
$       hush2
$       delete 'path'keyindex.tmp;*
$!!
$!!     Run the indexing program
$!!
$       hush1
$       hush2
$       run/nodebug index
$!!
$!!     And clean up any temporary files on the way out.
$!!
$       hush1
$       hush2
$       delete 'path'keyindex.tmp;*
$       hush1
$       hush2
$       delete 'path'keyfile.tmp;*
$!!
$!!     The key loading has been completed.  Control is transfered
$!!     back to the calling program
$!!
$!!     Signal to the user and return
$       ws bell
$       ws line
$       ws line
$       ws "The keyload process has finished"
$!!
$       exit
```

```
$!!     ********************  LOGICALS.COM  ********************
$!!
$!!     This .COM file contains all the process logical definitions
$!!     that are used by the key management code.
$!!
$       PATH == "UD:[CRYPTO.KEYMGR]"
$       WS == "WRITE SYS$OUTPUT"
$       BLANK_LINE == ""
$       BELL == ""
$       ERASE == ""
$       START == ""
$       LINE == ""
$       hush1 == "assign/user nl:  sys$error"
$       hush2 == "assign/user nl:  sys$output"
$       pause == "wait 00:00:05"
$       error_cond == "OFF"
```

Listings of the crypto-services programs. The programs actually implementing the key utilization and the DES processing are

GETKEY.ØF
FETCHKEY.QF
CBC_CYPHER.FOR
X_OR.FOR
PDES

```
c
c
c       ********************   getkey.qf   ********************
c
c
c       This subroutine is one of the pair of the key service routines
c       prepared to support the WBCN DES encryption processing.
c
c       This subroutine returns a key and key pointer when it is called
c       with the origination node and destination node as the arguements.
c       The routine takes care of the rotation of the key through the
c       selection of available keys.
c
c
        subroutine getkey (orig_site, dest_site, fkey_ptr, fkey, ferror)
c
##      declare
c
c       The declared fortran variables used in the routine are:
c
c               orig_site == the source gateway node
c
c               dest_site == the destination gateway node
c
c               fkey_ptr  == the integer pointer that identifies the key
c
c               fkey      == the key in character form
c
c               ferror    == the error flag.  Set to 10 if the
c                            dest_site is not included in the key table
c
c               fnode     == verification parameter used to confirm that
c                            the destinition is reachable
c
c               fperiod   == the current key period
c
c
c
##      character*20    orig_site, dest_site, fnode
##      character*16    fkey
##      integer         fkey_ptr, iptr, ferror, detect
##      character*3     fcond, new, old
##      integer*4       fperiod
c
c
c
        data    new     / 'new' /
        data    old     / 'old' /
c
c
##      ingres keyuser
c
c
c       Set up the table reference pointers
c
##      range of kt is keytable
##      range of nt is nodetable
```

```
##      range of pd is prdtable
c
c       First check to see if the destination keys are available to this node
c
        ferror = 0
        detect = 0
c
##      retrieve (fnode = nt.dest_node) where nt.dest_node = dest_site
##      {
        detect = detect + 1
##      }
c
c       If detect is still zero then the dest_site is not reachable
c       that being the case set the error flag and retrun
c
c
        if (detect .eq. 0) then
                ferror = 10
##              exit
                go to 20
        end if
c
c       Otherwise continue toward getting the key
c
c       The next step is to get the current key period
c
c
##      retrieve (fperiod= max (pd.period
##                      where "today" >= pd.period_date  ))
c
c
c       Put in a failsafe reference
c
        if (fperiod .eq. 0 ) fperiod = 1
c
c       It turns out that since the old keys are not purged from the
c       key table the reference to first period keys will still be
c       valid.
c
10      continue
c
c       Do the initial retrieve on the key table.  In addition check for
c       a completely used key set.  If the key set is completely used reset
c       it and come back here to start again.
c
c
        iptr = 0
c
##      retrieve (fkey_ptr = kt.keyptr, fkey = kt.key)
##         where kt.orig_node = orig_site
##           and kt.dest_node = dest_site
##           and kt.cond      = new
##           and kt.period    = fperiod
##      {
c
        iptr = iptr + 1
c
```

```
##      }
c
c
c       Check to see if any keys were returned.  If so reset the use
c       pointer and return.  Otherwise reset the table and start again
c
c
        if(iptr .gt. 0 )  then
c
##              replace kt(cond = old) where kt.keyptr = fkey_ptr
##              exit
                return
c
        else
c
##              replace kt(cond = new, nuse = kt.nuse + 1)
##                where kt.orig_node = orig_site
##                  and kt.dest_node = dest_site
##                  and kt.period    = fperiod
c
        end if
c
c       The table has been reset, now go get a key
c
        go to 10
c
c       This is the branch point if a known error is detected
c
20      continue
c
        return
c
        end
```

```
c
c       ********************   fetchkey.qf   ********************
c
c
c       This subroutine returns the key that corresponds to a given pointer.
c       The idea is that when the WBCN initiates a connection the subroutine
c       getkey  will be called to acquire the session key and key pointer.
c       Then as the connection is established the origination node sends the
c       key pointer to the destination node.  the destination node makes a
c       call to this subroutine with the pointer as the arguement to
c       retrieve the key.  This scheme requires the origination node to
c       make all the key requirements decisions before it originally
c       acquires the key and pointer.  However with this scheme no discussion
c       is required between the origination and destination nodes over
c       the correct key identification.
c
c
        subroutine fetchkey (fpointer, fkey, ferror)
c
##      declare
c
c
##      character*16        fkey
##      integer*4           fpointer
##      integer             ferror, detect
c
c
c
##      ingres keymgr
c
##      range of pt is ptrtable
c
c       There is a modest error detection procedure built into this.  If
c       this key fetch does not return a key then the flag ferror is set to 10.
c       Otherwise it is returned as 0.
c
c
        ferror = 0
        detect = 0
c
c
##      retrieve (fkey = pt.key) where pt.keyptr = fpointer
##      {
        detect = detect + 1
##      }
c
c
        if (detect .eq. 0 )  ferror = 10
c
        return
c
        end
```

```
c       ********************   cbc_cypher.for   ********************
c
c       This subroutine accesses the cypher block chaining mode of the
c       DES module. The routine is called with the cypher direction flag,
c       the process key, the initial vector, the input buffer, the output
c       buffer, the buffer length, and the returned output iv.
c
c       There are a few things that the user of this subroutine should
c       consider.  First, the cypher processor expects to process an
c       integral number of 64 bit clear text words.  This will usually
c       not be too much of a problem however it must be considered.
c       Be sure that the input buffer has been padded out to the full
c       64 bit boundary before calling the module.
c
c
c       ************************************************************************
c       ************************************************************************
c
        subroutine cbc_cypher(cflag, key, keyptr, buffin, buffot, len )
c
c
c
c
c       input
c
c               cflag  ==  char*1 == the cypher direction indicator:  'E'
c                                    for encrypt, 'D' for decrypt
c
c               key    ==  real*8 == the clear-text cypher key 64 bits
c                                    initializes as a hexadecimal constant
c
c               buffin ==  real*8 == the input string configured as a string
c                                    of 64 bit words
c
c               len    ==  intgr  == the number of 64 bit words to be
c                                    processed
c
c               keyptr ==  intgr  == the key pointer used to create an iv
c
c
c
c       output
c
c               buffot == real*8 == the cyphered result of applying the cflag
c                                   cypher direction with the variable key
c                                   and initial vector iv to the contents of
c                                   buffin
c
c
c       ************************************************************************
c
c
c
c
c
c
c       define and initialize the local paramaters
```

```
c
c
      character*1       cflag
c
      real*8            buffin(len),  buffot(len),   key,   iv,  ivout
c
      integer           encrypt,   decrypt,   cbc_mode,   len
c
      integer*4         keyptr
c
      data              encrypt,  decrypt       / 1,   2   /
      data              cbc_mode                / 1        /
c
c
c     End of the definitions
c
c
c
c     Start the processing
c
c     Create the iv used for this mode of the DES processor
c
      call x_or  ( key,  keyptr, iv  )
c
c     Decide which way to run the cypher processor
c
      if     (cflag .eq.  'E' )  then
c
c
c              run the cypher in the encrypt mode
c
c              set the mode to cbc
c
               call set_mode ( cbc_mode )
c
c              set the process direction to encrypt
c
               call set_dir ( encrypt )
c
c              load the key and iv
c
               call load_key ( key,  iv )
c
c              stream the buffer through the processor and recover the iv
c
               call pdes( buffin, buffot, len, ivout )
c
c              this ends the the encrypt pass now retrun
c
c
      else if (cflag .eq. 'D' )   then

c
c              run the cypher in the decrypt mode
c
c              set the mode to cbc
```

```
c
                call set_mode ( cbc_mode )
c
c               set the process direction to decrypt
c
                call set_dir ( decrypt )
c
c               load the key and iv
c
                call load_key ( key,  iv )
c
c               stream the buffer through the processor and recover the iv
c
                call pdes( buffin, buffot, len, ivout )
c
c               this ends the decrypt pass now retrun
c
c
        else
c
c               there is a call error
c
c               any desirable error handling should /could be put here
c
                return

c
        end if
c
c
        return
c
c
        end
c
c
```

```
c       ********************   x_or.for   ********************
c
c       This subroutine is designed to provide the x-or of two 64-bit
c       strings that are passed to it.
c
        subroutine  x_or (inf, ins, iout)
c
c
c       the arguments for this subroutine are:
c
c                       inf == input first == the first of the two 64 bit
c                                             quad words to be x_or-ed
c
c                       ins == input second == the second of the two 64 bit
c                                              quad words to be x_or-ed
c
c                       iout == output string == the result output
c
c       The arguements must be integers from here down since the intrinsic
c       function needs and returns integers
c
c
        integer*4  inf(2),  ins(2),  iout(2)
c
c       They are dimensioned (2)  to be able to hold all 64 bits
c
c
        iout (1) = ieor ( inf(1), ins(1) )
        iout (2) = ieor ( inf(2), ins(2) )
c
        return
c
        end
```

```
        .TITLE  PDES - PUBLIC DATA ENCRYPTION STANDARD
;
; AUTHOR:                RICH BELLES
;
; DATE:                    8/21/84
;
;
; DEFINE SELECTION FUNCTIONS
;
S_1A::  .WORD ^X0014,^X0000,^X0000,^X0014,^X0014,^X0000,^X0000,^X0000
        .WORD ^X0000,^X0014,^X0014,^X0000,^X0014,^X0014,^X0014,^X0000
        .WORD ^X0000,^X0014,^X0014,^X0000,^X0000,^X0014,^X0014,^X0014
        .WORD ^X0000,^X0014,^X0014,^X0000,^X0000,^X0000,^X0000,^X0014
        .WORD ^X0000,^X0014,^X0000,^X0014,^X0014,^X0014,^X0014,^X0000
        .WORD ^X0014,^X0000,^X0000,^X0014,^X0000,^X0000,^X0014,^X0000
        .WORD ^X0014,^X0000,^X0014,^X0014,^X0014,^X0000,^X0000,^X0014
        .WORD ^X0000,^X0014,^X0014,^X0000,^X0000,^X0000,^X0000,^X0014
;
S_2A::  .WORD ^X2000,^X2000,^X0000,^X0000,^X0000,^X0000,^X2000,^X2000
        .WORD ^X2000,^X2000,^X2000,^X2000,^X2000,^X0000,^X0000,^X2000
        .WORD ^X0000,^X0000,^X2000,^X0000,^X2000,^X0000,^X0000,^X2000
        .WORD ^X0000,^X2000,^X0000,^X0000,^X0000,^X2000,^X2000,^X0000
        .WORD ^X0000,^X0000,^X2000,^X0000,^X2000,^X2000,^X2000,^X0000
        .WORD ^X2000,^X2000,^X0000,^X2000,^X0000,^X0000,^X0000,^X2000
        .WORD ^X0000,^X2000,^X0000,^X2000,^X0000,^X2000,^X2000,^X0000
        .WORD ^X0000,^X0000,^X2000,^X0000,^X2000,^X2000,^X2000,^X0000
;
S_3A::  .WORD ^X0000,^X0080,^X0000,^X0080,^X0080,^X0000,^X0000,^X0080
        .WORD ^X0000,^X0080,^X0080,^X0000,^X0080,^X0000,^X0080,^X0000
        .WORD ^X0080,^X0000,^X0080,^X0000,^X0000,^X0080,^X0080,^X0000
        .WORD ^X0080,^X0000,^X0000,^X0080,^X0000,^X0080,^X0000,^X0080
        .WORD ^X0080,^X0080,^X0000,^X0000,^X0000,^X0080,^X0080,^X0000
        .WORD ^X0000,^X0000,^X0080,^X0080,^X0080,^X0000,^X0000,^X0080
        .WORD ^X0080,^X0000,^X0080,^X0080,^X0000,^X0000,^X0000,^X0080
        .WORD ^X0080,^X0080,^X0000,^X0080,^X0000,^X0000,^X0080,^X0000
;
S_4A::  .WORD ^X4002,^X4000,^X4000,^X0000,^X0002,^X4002,^X4002,^X4000
        .WORD ^X0000,^X0002,^X0002,^X4002,^X4000,^X0000,^X0002,^X4002
        .WORD ^X4000,^X0000,^X0002,^X4002,^X0000,^X0002,^X4000,^X0000
        .WORD ^X4002,^X4000,^X0000,^X0002,^X0000,^X0002,^X4002,^X4000
        .WORD ^X0002,^X4002,^X0002,^X4002,^X4000,^X0000,^X0000,^X0002
        .WORD ^X0000,^X0002,^X4002,^X4000,^X4002,^X4000,^X4000,^X0000
        .WORD ^X4002,^X4000,^X4000,^X0000,^X4002,^X4000,^X0002,^X4002
        .WORD ^X4000,^X0000,^X0002,^X4002,^X0000,^X0002,^X0000,^X0002
;
S_5A::  .WORD ^X0000,^X0028,^X0028,^X1028,^X0000,^X0000,^X1000,^X0028
        .WORD ^X1000,^X0000,^X0028,^X1000,^X1028,^X1028,^X0000,^X1000
        .WORD ^X0028,^X1000,^X1000,^X0000,^X1000,^X1028,^X1028,^X0028
        .WORD ^X1028,^X1000,^X0000,^X1028,^X0028,^X0028,^X1028,^X0000
        .WORD ^X0000,^X1028,^X0000,^X0028,^X1000,^X0028,^X1028,^X1000
        .WORD ^X0028,^X1000,^X1028,^X0028,^X1000,^X0000,^X0028,^X1028
        .WORD ^X1028,^X0000,^X1028,^X1028,^X0028,^X0000,^X1000,^X1028
        .WORD ^X0000,^X0028,^X1000,^X0000,^X0000,^X1000,^X0028,^X1000
;
S_6A::  .WORD ^X0A00,^X0A01,^X0000,^X0A01,^X0A01,^X0000,^X0A01,^X0001
        .WORD ^X0A00,^X0001,^X0001,^X0A00,^X0001,^X0A00,^X0A00,^X0000
        .WORD ^X0000,^X0001,^X0A00,^X0000,^X0001,^X0A00,^X0000,^X0A01
```

```
        .WORD ^X0A01,^X0000,^X0001,^X0A01,^X0000,^X0001,^X0A01,^X0A00
        .WORD ^X0A00,^X0000,^X0A01,^X0001,^X0A01,^X0001,^X0000,^X0A00
        .WORD ^X0001,^X0A00,^X0A00,^X0000,^X0A00,^X0A01,^X0001,^X0A01
        .WORD ^X0001,^X0A01,^X0000,^X0A01,^X0000,^X0000,^X0A01,^X0001
        .WORD ^X0000,^X0001,^X0A00,^X0000,^X0A01,^X0A00,^X0001,^X0A00
;
S_7A::  .WORD ^X0000,^X8040,^X8040,^X0000,^X0000,^X8040,^X8000,^X0040
        .WORD ^X8040,^X0000,^X0000,^X8040,^X8000,^X0040,^X8040,^X8000
        .WORD ^X0040,^X8000,^X8000,^X0040,^X8040,^X0040,^X0040,^X8000
        .WORD ^X0040,^X0000,^X8000,^X8040,^X0000,^X8000,^X0040,^X0000
        .WORD ^X0040,^X0000,^X0000,^X8040,^X8040,^X8040,^X8040,^X8000
        .WORD ^X8000,^X0040,^X0040,^X0000,^X0040,^X8000,^X8000,^X0040
        .WORD ^X8000,^X8040,^X8040,^X0040,^X0000,^X0000,^X8000,^X8040
        .WORD ^X0000,^X8000,^X0040,^X0000,^X8040,^X0040,^X0000,^X8000
;
S_8A::  .WORD ^X0500,^X0000,^X0000,^X0500,^X0500,^X0500,^X0000,^X0500
        .WORD ^X0000,^X0500,^X0500,^X0000,^X0500,^X0000,^X0000,^X0000
        .WORD ^X0500,^X0500,^X0500,^X0000,^X0000,^X0000,^X0500,^X0500
        .WORD ^X0000,^X0000,^X0000,^X0500,^X0500,^X0500,^X0000,^X0000
        .WORD ^X0000,^X0000,^X0500,^X0000,^X0000,^X0500,^X0000,^X0000
        .WORD ^X0500,^X0000,^X0500,^X0500,^X0500,^X0500,^X0000,^X0500
        .WORD ^X0000,^X0500,^X0000,^X0500,^X0500,^X0500,^X0500,^X0000
        .WORD ^X0500,^X0000,^X0000,^X0000,^X0000,^X0000,^X0500,^X0500
;
S_1B::  .LONG ^X01404000,^X00000000,^X01400000,^X01404004
        .LONG ^X01400004,^X01404004,^X00000004,^X01400000
        .LONG ^X00004000,^X01404000,^X01404004,^X00004000
        .LONG ^X00004004,^X01400004,^X00000000,^X00000004
        .LONG ^X00004004,^X00004000,^X00004000,^X01404000
        .LONG ^X01404000,^X01400000,^X01400000,^X00004004
        .LONG ^X01400004,^X00000004,^X00000004,^X01400004
        .LONG ^X00000000,^X00004004,^X01404004,^X00000000
        .LONG ^X01400000,^X01404004,^X00000004,^X01400000
        .LONG ^X01404000,^X00000000,^X00000000,^X00004000
        .LONG ^X01400004,^X01400000,^X01404000,^X00000004
        .LONG ^X00004000,^X00000004,^X00004004,^X01404004
        .LONG ^X01404004,^X01400004,^X01400000,^X00004004
        .LONG ^X00000004,^X00004004,^X01404004,^X01404000
        .LONG ^X00004004,^X00004000,^X00004000,^X00000000
        .LONG ^X01400004,^X01404000,^X00000000,^X01400004
;
S_2B::  .LONG ^X502000A0,^X00200000,^X00200000,^X502000A0
        .LONG ^X50000000,^X000000A0,^X500000A0,^X002000A0
        .LONG ^X000000A0,^X502000A0,^X50200000,^X00000000
        .LONG ^X00200000,^X50000000,^X000000A0,^X500000A0
        .LONG ^X50200000,^X500000A0,^X002000A0,^X00000000
        .LONG ^X00000000,^X00200000,^X502000A0,^X50000000
        .LONG ^X500000A0,^X000000A0,^X00000000,^X50200000
        .LONG ^X002000A0,^X50200000,^X50000000,^X002000A0
        .LONG ^X00000000,^X502000A0,^X500000A0,^X50000000
        .LONG ^X002000A0,^X50000000,^X50200000,^X00200000
        .LONG ^X50000000,^X00200000,^X000000A0,^X502000A0
        .LONG ^X502000A0,^X000000A0,^X00200000,^X00000000
        .LONG ^X002000A0,^X50200000,^X50000000,^X000000A0
        .LONG ^X500000A0,^X002000A0,^X000000A0,^X500000A0
        .LONG ^X50200000,^X00000000,^X00200000,^X002000A0
        .LONG ^X00000000,^X500000A0,^X502000A0,^X50200000
```

```
;
S_3B::  .LONG ^X00002808,^X02802800,^X00000000,^X02800008
        .LONG ^X00002800,^X00000000,^X02802808,^X00002800
        .LONG ^X02800008,^X00000008,^X00000008,^X02800000
        .LONG ^X02802808,^X02800008,^X02800000,^X00002808
        .LONG ^X00000000,^X00000008,^X02802800,^X00002800
        .LONG ^X02802800,^X02800000,^X02800008,^X02802808
        .LONG ^X00002808,^X02802800,^X02800000,^X00002808
        .LONG ^X00000008,^X02802808,^X00002800,^X00000000
        .LONG ^X02802800,^X00000000,^X02800008,^X00002808
        .LONG ^X02800000,^X02802800,^X00002800,^X00000000
        .LONG ^X00002800,^X02800008,^X02802808,^X00002800
        .LONG ^X00000008,^X00002800,^X00000000,^X02800008
        .LONG ^X00002808,^X02800000,^X00000000,^X02802808
        .LONG ^X00000008,^X02802808,^X02802800,^X00000008
        .LONG ^X02800000,^X00002808,^X00002808,^X02800000
        .LONG ^X02802808,^X00000008,^X02800008,^X02802800
;
S_4B::  .LONG ^X000A0001,^X000A0201,^X000A0201,^X00000200
        .LONG ^X000A0200,^X00000201,^X00000001,^X000A0001
        .LONG ^X00000000,^X000A0000,^X000A0000,^X000A0201
        .LONG ^X00000201,^X00000000,^X00000200,^X00000001
        .LONG ^X00000001,^X000A0000,^X00000000,^X000A0001
        .LONG ^X00000200,^X00000000,^X000A0001,^X000A0200
        .LONG ^X00000201,^X00000001,^X000A0200,^X00000200
        .LONG ^X000A0000,^X000A0200,^X000A0201,^X00000201
        .LONG ^X00000200,^X00000001,^X000A0000,^X000A0201
        .LONG ^X00000201,^X00000000,^X00000000,^X000A0000
        .LONG ^X000A0200,^X00000200,^X00000201,^X00000001
        .LONG ^X000A0001,^X000A0201,^X000A0201,^X00000200
        .LONG ^X000A0201,^X00000201,^X00000001,^X000A0000
        .LONG ^X00000001,^X000A0001,^X000A0200,^X00000201
        .LONG ^X000A0001,^X000A0200,^X00000000,^X000A0001
        .LONG ^X00000200,^X00000000,^X000A0000,^X000A0200
;
S_5B::  .LONG ^X00001400,^X08001400,^X08000000,^X00001400
        .LONG ^X08000000,^X00001400,^X00000000,^X08000000
        .LONG ^X08001400,^X08000000,^X00001400,^X08001400
        .LONG ^X00001400,^X08000000,^X08001400,^X00000000
        .LONG ^X00000000,^X08000000,^X08000000,^X00000000
        .LONG ^X00001400,^X08001400,^X08001400,^X00001400
        .LONG ^X08000000,^X00001400,^X00000000,^X00000000
        .LONG ^X08001400,^X00000000,^X00000000,^X08001400
        .LONG ^X08000000,^X00001400,^X00001400,^X00000000
        .LONG ^X00000000,^X08000000,^X00001400,^X08001400
        .LONG ^X00001400,^X00000000,^X08000000,^X08001400
        .LONG ^X08001400,^X00001400,^X00000000,^X08000000
        .LONG ^X08001400,^X08001400,^X00000000,^X08001400
        .LONG ^X08000000,^X00000000,^X08000000,^X00000000
        .LONG ^X08001400,^X00001400,^X00001400,^X08000000
        .LONG ^X00000000,^X08000000,^X08001400,^X00001400
;
S_6B::  .LONG ^X00000050,^X00000000,^X00100000,^X00100050
        .LONG ^X00000000,^X00000050,^X00100050,^X00000000
        .LONG ^X00100000,^X00100050,^X00000000,^X00000050
        .LONG ^X00000050,^X00100000,^X00000000,^X00100050
        .LONG ^X00000000,^X00000050,^X00100050,^X00100000
```

```
        .LONG ^X00100000,^X00100050,^X00000050,^X00000050
        .LONG ^X00000050,^X00000000,^X00100050,^X00100000
        .LONG ^X00100050,^X00100000,^X00100000,^X00000000
        .LONG ^X00100000,^X00000050,^X00000050,^X00100000
        .LONG ^X00100050,^X00000000,^X00100050,^X00000050
        .LONG ^X00000000,^X00100000,^X00000000,^X00100050
        .LONG ^X00000050,^X00100050,^X00100000,^X00000000
        .LONG ^X00100050,^X00100000,^X00000000,^X00000050
        .LONG ^X00000050,^X00100000,^X00000000,^X00100050
        .LONG ^X00100000,^X00000050,^X00100050,^X00000000
        .LONG ^X00100000,^X00000000,^X00000050,^X00100050
;
S_7B::  .LONG ^XA0000000,^XA0000002,^X00008002,^X00000000
        .LONG ^X00008000,^X00008002,^XA0008002,^XA0008000
        .LONG ^XA0008002,^XA0000000,^X00000000,^X00000002
        .LONG ^X00000002,^X00000000,^XA0000002,^X00008002
        .LONG ^X00008000,^XA0008002,^XA0000002,^X00008000
        .LONG ^X00000002,^XA0000000,^XA0008000,^XA0000002
        .LONG ^XA0000000,^X00008000,^X00008002,^XA0008002
        .LONG ^XA0008000,^X00000002,^X00000000,^XA0008000
        .LONG ^X00000000,^XA0008000,^XA0000000,^X00008002
        .LONG ^X00008002,^XA0000002,^XA0000002,^X00000002
        .LONG ^XA0000002,^X00000000,^X00008000,^XA0000000
        .LONG ^XA0008000,^X00008002,^XA0008002,^XA0008000
        .LONG ^X00008002,^X00000002,^XA0008002,^XA0000000
        .LONG ^XA0008000,^X00000000,^X00000002,^XA0008002
        .LONG ^X00000000,^XA0008002,^XA0000000,^X00008000
        .LONG ^X00000002,^X00008000,^X00008000,^XA0000002
;
S_8B::  .LONG ^X00050100,^X00050000,^X04000000,^X04050100
        .LONG ^X00000000,^X00050100,^X00000100,^X00000000
        .LONG ^X04000100,^X04000000,^X04050100,^X04050000
        .LONG ^X04050000,^X04050100,^X00050000,^X00000100
        .LONG ^X04000000,^X00000100,^X00050000,^X00050100
        .LONG ^X04050000,^X04000100,^X04000100,^X04050000
        .LONG ^X00050100,^X00000000,^X00000000,^X04000100
        .LONG ^X00000100,^X00050000,^X04050100,^X04000000
        .LONG ^X04050100,^X04000000,^X04050000,^X00050000
        .LONG ^X00000100,^X04000100,^X00050000,^X04050100
        .LONG ^X00050000,^X00000100,^X00000100,^X04000000
        .LONG ^X04000100,^X00000000,^X04000000,^X00050100
        .LONG ^X00000000,^X04050100,^X04000100,^X00000100
        .LONG ^X04000000,^X00050000,^X00050100,^X00000000
        .LONG ^X04050100,^X04050000,^X04050000,^X00050100
        .LONG ^X00050100,^X04000100,^X00000000,^X04050000
;
K_EYS:: .BLKB   96
I_VEC:: .BLKB   8
M_ODE:: .BLKB   4
D_IR::  .BLKB   4
I_ADD:: .BLKB   4
O_ADD:: .BLKB   4
L_EN::  .BLKB   4
SREG:   .BLKB   8
;
;
; BEGIN EXECUTABLE CODE
```

```
;
;       ************************************************
;       *                                              *
;       *               ENTRY POINT LOAD_KEY           *
;       *                                              *
;       ************************************************
;
; CALLING SEQUENCE:
;
;                   CALL LOAD_KEY (IKEY,IVEC)
;
; PURPOSE:
;
;       LOAD KEY AND INITIAL VECTOR
;
; ARGUMENTS:
;
;       IKEY    :  64-BIT KEY FOR USE IN ENCIPHERING/DECIPHERING.
;
;       IVEC    :  64-BIT INITIAL VECTOR.
;
;
; OFFSETS FOR ARGUMENT LIST
;
        IKEY=4
        IVEC=8
;
        .ENTRY LOAD_KEY ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
;
; CALCULATE KEYS
;
        MOVQ    @IVEC(AP),I_VEC
        MOVQ    @IKEY(AP),R2            ; R0,R1 = IKEY
        MOVL    R2,R1
        MOVL    R3,R0
;
        MOVQ    R12,SREG                ; SAVE REGISTERS R12 & R13
;
        CLRQ    R2
        CLRQ    R4
        CLRQ    R6
        CLRQ    R8
        CLRQ    R10
        CLRQ    R12
;
        BBC     #31,R0,100$
        BBSS    #12,R2,.+1
        BBSS    #6,R3,.+1
        BBSS    #16,R5,.+1
        BBSS    #24,R7,.+1
        BBSS    #24,R8,.+1
        BBSS    #31,R10,.+1
        BBSS    #28,R11,.+1
100$:   BBC     #30,R0,110$
        BBSS    #23,R2,.+1
        BBSS    #15,R3,.+1
        BBSS    #17,R5,.+1
```

```
        BBSS    #4,R6,.+1
        BBSS    #14,R8,.+1
        BBSS    #6,R9,.+1
        BBSS    #16,R11,.+1
        BBSS    #24,FP,.+1
110$:   BBC     #29,R0,120$
        BBSS    #19,R2,.+1
        BBSS    #10,R5,.+1
        BBSS    #13,R8,.+1
        BBSS    #15,R9,.+1
        BBSS    #17,R11,.+1
        BBSS    #4,R12,.+1
120$:   BBC     #28,R0,130$
        BBSS    #2,R2,.+1
        BBSS    #6,R5,.+1
        BBSS    #3,R7,.+1
        BBSS    #28,R9,.+1
        BBSS    #5,R10,.+1
        BBSS    #27,R12,.+1
        BBSS    #4,FP,.+1
130$:   BBC     #27,R0,140$
        BBSS    #31,R3,.+1
        BBSS    #3,R4,.+1
        BBSS    #28,R6,.+1
        BBSS    #5,R7,.+1
        BBSS    #27,R9,.+1
        BBSS    #4,R10,.+1
        BBSS    #0,R11,.+1
140$:   BBC     #26,R0,150$
        BBSS    #4,R4,.+1
        BBSS    #0,R5,.+1
        BBSS    #18,R9,.+1
        BBSS    #7,R10,.+1
        BBSS    #16,R12,.+1
        BBSS    #17,FP,.+1
150$:   BBC     #25,R0,160$
        BBSS    #7,R4,.+1
        BBSS    #16,R6,.+1
        BBSS    #17,R7,.+1
        BBSS    #24,R9,.+1
        BBSS    #6,R11,.+1
        BBSS    #3,FP,.+1
160$:   BBC     #23,R0,170$
        BBSS    #22,R2,.+1
        BBSS    #10,R3,.+1
        BBSS    #25,R5,.+1
        BBSS    #11,R6,.+1
        BBSS    #11,R8,.+1
        BBSS    #19,R11,.+1
        BBSS    #5,R12,.+1
170$:   BBC     #22,R0,180$
        BBSS    #31,R2,.+1
        BBSS    #25,R4,.+1
        BBSS    #29,R5,.+1
        BBSS    #12,R8,.+1
        BBSS    #10,R9,.+1
        BBSS    #25,R11,.+1
```

```
        BBSS    #11,R12,.+1
180$:   BBC     #21,R0,190$
        BBSS    #5,R3,.+1
        BBSS    #18,R5,.+1
        BBSS    #14,R6,.+1
        BBSS    #23,R8,.+1
        BBSS    #25,R10,.+1
        BBSS    #29,R11,.+1
190$:   BBC     #20,R0,200$
        BBSS    #6,R4,.+1
        BBSS    #31,R6,.+1
        BBSS    #10,R7,.+1
        BBSS    #3,R8,.+1
        BBSS    #14,R10,.+1
        BBSS    #23,FP,.+1
200$:   BBC     #19,R0,210$
        BBSS    #19,R3,.+1
        BBSS    #10,R4,.+1
        BBSS    #3,R5,.+1
        BBSS    #14,R7,.+1
        BBSS    #23,R10,.+1
        BBSS    #25,R12,.+1
        BBSS    #20,FP,.+1
210$:   BBC     #18,R0,220$
        BBSS    #20,R3,.+1
        BBSS    #23,R4,.+1
        BBSS    #25,R6,.+1
        BBSS    #20,R7,.+1
        BBSS    #13,R10,.+1
        BBSS    #5,R11,.+1
        BBSS    #1,FP,.+1
220$:   BBC     #17,R0,230$
        BBSS    #23,R3,.+1
        BBSS    #13,R4,.+1
        BBSS    #5,R5,.+1
        BBSS    #1,R7,.+1
        BBSS    #2,R8,.+1
        BBSS    #6,R10,.+1
        BBSS    #31,R12,.+1
        BBSS    #10,FP,.+1
230$:   BBC     #15,R0,240$
        BBSS    #26,R2,.+1
        BBSS    #0,R3,.+1
        BBSS    #8,R5,.+1
        BBSS    #8,R6,.+1
        BBSS    #15,R8,.+1
        BBSS    #12,R9,.+1
        BBSS    #26,FP,.+1
240$:   BBC     #14,R0,250$
        BBSS    #9,R2,.+1
        BBSS    #1,R3,.+1
        BBSS    #20,R5,.+1
        BBSS    #30,R7,.+1
        BBSS    #22,R8,.+1
        BBSS    #0,R9,.+1
        BBSS    #8,R11,.+1
        BBSS    #8,R12,.+1
```

```
250$:   BBC     #13,R0,260$
        BBSS    #21,R2,.+1
        BBSS    #26,R4,.+1
        BBSS    #29,R7,.+1
        BBSS    #31,R8,.+1
        BBSS    #1,R9,.+1
        BBSS    #20,R11,.+1
        BBSS    #30,FP,.+1
260$:   BBC     #12,R0,270$
        BBSS    #22,R3,.+1
        BBSS    #22,R4,.+1
        BBSS    #19,R6,.+1
        BBSS    #12,R7,.+1
        BBSS    #21,R9,.+1
        BBSS    #11,R10,.+1
        BBSS    #20,R12,.+1
        BBSS    #16,FP,.+1
270$:   BBC     #11,R0,280$
        BBSS    #26,R3,.+1
        BBSS    #12,R4,.+1
        BBSS    #21,R6,.+1
        BBSS    #11,R7,.+1
        BBSS    #20,R9,.+1
        BBSS    #16,R10,.+1
        BBSS    #2,FP,.+1
280$:   BBC     #10,R0,290$
        BBSS    #7,R2,.+1
        BBSS    #16,R4,.+1
        BBSS    #2,R7,.+1
        BBSS    #23,R9,.+1
        BBSS    #0,R10,.+1
        BBSS    #1,R11,.+1
        BBSS    #8,FP,.+1
290$:   BBC     #9,R0,300$
        BBSS    #29,R3,.+1
        BBSS    #0,R4,.+1
        BBSS    #1,R5,.+1
        BBSS    #8,R7,.+1
        BBSS    #22,R10,.+1
        BBSS    #19,R12,.+1
        BBSS    #12,FP,.+1
300$:   BBC     #7,R0,310$
        BBSS    #16,R2,.+1
        BBSS    #9,R3,.+1
        BBSS    #27,R5,.+1
        BBSS    #27,R7,.+1
        BBSS    #3,R9,.+1
        BBSS    #21,R11,.+1
        BBSS    #2,R12,.+1
310$:   BBC     #6,R0,320$
        BBSS    #17,R2,.+1
        BBSS    #13,R3,.+1
        BBSS    #28,R7,.+1
        BBSS    #26,R8,.+1
        BBSS    #9,R9,.+1
        BBSS    #27,R11,.+1
        BBSS    #27,FP,.+1
```

```
320$:   BBC     #5,R0,330$
        BBSS    #10,R2,.+1
        BBSS    #2,R3,.+1
        BBSS    #30,R5,.+1
        BBSS    #7,R6,.+1
        BBSS    #9,R8,.+1
        BBSS    #13,R9,.+1
        BBSS    #28,FP,.+1
330$:   BBC     #4,R0,340$
        BBSS    #6,R2,.+1
        BBSS    #15,R4,.+1
        BBSS    #26,R6,.+1
        BBSS    #19,R7,.+1
        BBSS    #30,R9,.+1
        BBSS    #7,R11,.+1
        BBSS    #9,FP,.+1
340$:   BBC     #3,R0,350$
        BBSS    #28,R3,.+1
        BBSS    #19,R4,.+1
        BBSS    #30,R6,.+1
        BBSS    #7,R8,.+1
        BBSS    #9,R10,.+1
        BBSS    #4,R11,.+1
350$:   BBC     #2,R0,360$
        BBSS    #0,R2,.+1
        BBSS    #9,R4,.+1
        BBSS    #4,R5,.+1
        BBSS    #29,R9,.+1
        BBSS    #21,R10,.+1
        BBSS    #17,R12,.+1
        BBSS    #18,FP,.+1
360$:   BBC     #1,R0,370$
        BBSS    #16,R3,.+1
        BBSS    #21,R4,.+1
        BBSS    #17,R6,.+1
        BBSS    #18,R7,.+1
        BBSS    #22,R9,.+1
        BBSS    #15,R10,.+1
        BBSS    #26,R12,.+1
        BBSS    #19,FP,.+1
370$:   BBC     #31,R1,380$
        BBSS    #25,R2,.+1
        BBSS    #24,R4,.+1
        BBSS    #24,R5,.+1
        BBSS    #31,R7,.+1
        BBSS    #28,R8,.+1
        BBSS    #10,R11,.+1
380$:   BBC     #30,R1,390$
        BBSS    #29,R2,.+1
        BBSS    #4,R3,.+1
        BBSS    #14,R5,.+1
        BBSS    #6,R6,.+1
        BBSS    #16,R8,.+1
        BBSS    #24,R10,.+1
        BBSS    #24,R11,.+1
        BBSS    #31,FP,.+1
390$:   BBC     #29,R1,400$
```

```
        BBSS    #18,R2,.+1
        BBSS    #13,R5,.+1
        BBSS    #15,R6,.+1
        BBSS    #17,R8,.+1
        BBSS    #4,R9,.+1
        BBSS    #14,R11,.+1
        BBSS    #6,R12,.+1
400$:   BBC     #28,R1,410$
        BBSS    #11,R2,.+1
        BBSS    #31,R4,.+1
        BBSS    #28,R5,.+1
        BBSS    #10,R8,.+1
        BBSS    #13,R11,.+1
        BBSS    #15,R12,.+1
410$:   BBC     #27,R1,420$
        BBSS    #3,R2,.+1
        BBSS    #5,R4,.+1
        BBSS    #27,R6,.+1
        BBSS    #4,R7,.+1
        BBSS    #0,R8,.+1
        BBSS    #18,R12,.+1
        BBSS    #7,FP,.+1
420$:   BBC     #26,R1,430$
        BBSS    #25,R3,.+1
        BBSS    #18,R6,.+1
        BBSS    #7,R7,.+1
        BBSS    #16,R9,.+1
        BBSS    #17,R10,.+1
        BBSS    #24,R12,.+1
430$:   BBC     #25,R1,440$
        BBSS    #5,R2,.+1
        BBSS    #17,R4,.+1
        BBSS    #24,R6,.+1
        BBSS    #6,R8,.+1
        BBSS    #3,R10,.+1
        BBSS    #28,R12,.+1
        BBSS    #5,FP,.+1
440$:   BBC     #23,R1,450$
        BBSS    #8,R2,.+1
        BBSS    #11,R3,.+1
        BBSS    #11,R5,.+1
        BBSS    #19,R8,.+1
        BBSS    #5,R9,.+1
        BBSS    #18,R11,.+1
        BBSS    #14,R12,.+1
450$:   BBC     #22,R1,460$
        BBSS    #20,R2,.+1
        BBSS    #12,R5,.+1
        BBSS    #10,R6,.+1
        BBSS    #25,R8,.+1
        BBSS    #11,R9,.+1
        BBSS    #11,R11,.+1
460$:   BBC     #21,R1,470$
        BBSS    #14,R3,.+1
        BBSS    #23,R5,.+1
        BBSS    #25,R7,.+1
        BBSS    #29,R8,.+1
```

```
        BBSS    #12,R11,.+1
        BBSS    #10,R12,.+1
470$:   BBC     #20,R1,480$
        BBSS    #15,R2,.+1
        BBSS    #19,R5,.+1
        BBSS    #5,R6,.+1
        BBSS    #18,R8,.+1
        BBSS    #14,R9,.+1
        BBSS    #23,R11,.+1
        BBSS    #25,FP,.+1
480$:   BBC     #19,R1,490$
        BBSS    #21,R3,.+1
        BBSS    #14,R4,.+1
        BBSS    #23,R7,.+1
        BBSS    #25,R9,.+1
        BBSS    #20,R10,.+1
        BBSS    #13,FP,.+1
490$:   BBC     #18,R1,500$
        BBSS    #20,R4,.+1
        BBSS    #13,R7,.+1
        BBSS    #5,R8,.+1
        BBSS    #1,R10,.+1
        BBSS    #2,R11,.+1
        BBSS    #6,FP,.+1
500$:   BBC     #17,R1,510$
        BBSS    #1,R2,.+1
        BBSS    #1,R4,.+1
        BBSS    #2,R5,.+1
        BBSS    #6,R7,.+1
        BBSS    #31,R9,.+1
        BBSS    #10,R10,.+1
        BBSS    #3,R11,.+1
        BBSS    #14,FP,.+1
510$:   BBC     #15,R1,520$
        BBSS    #27,R2,.+1
        BBSS    #8,R3,.+1
        BBSS    #15,R5,.+1
        BBSS    #12,R6,.+1
        BBSS    #26,R10,.+1
        BBSS    #29,FP,.+1
520$:   BBC     #14,R1,530$
        BBSS    #30,R4,.+1
        BBSS    #22,R5,.+1
        BBSS    #0,R6,.+1
        BBSS    #8,R8,.+1
        BBSS    #8,R9,.+1
        BBSS    #15,R11,.+1
        BBSS    #12,R12,.+1
530$:   BBC     #13,R1,540$
        BBSS    #30,R2,.+1
        BBSS    #29,R4,.+1
        BBSS    #31,R5,.+1
        BBSS    #1,R6,.+1
        BBSS    #20,R8,.+1
        BBSS    #30,R10,.+1
        BBSS    #22,R11,.+1
        BBSS    #0,R12,.+1
```

```
540$:   BBC     #12,R1,550$
        BBSS    #12,R3,.+1
        BBSS    #26,R7,.+1
        BBSS    #29,R10,.+1
        BBSS    #31,R11,.+1
        BBSS    #1,R12,.+1
550$:   BBC     #11,R1,560$
        BBSS    #30,R3,.+1
        BBSS    #11,R4,.+1
        BBSS    #20,R6,.+1
        BBSS    #16,R7,.+1
        BBSS    #2,R10,.+1
        BBSS    #23,R12,.+1
        BBSS    #0,FP,.+1
560$:   BBC     #10,R1,570$
        BBSS    #4,R2,.+1
        BBSS    #2,R4,.+1
        BBSS    #23,R6,.+1
        BBSS    #0,R7,.+1
        BBSS    #1,R8,.+1
        BBSS    #8,R10,.+1
        BBSS    #22,FP,.+1
570$:   BBC     #9,R1,580$
        BBSS    #17,R3,.+1
        BBSS    #8,R4,.+1
        BBSS    #22,R7,.+1
        BBSS    #19,R9,.+1
        BBSS    #12,R10,.+1
        BBSS    #21,R12,.+1
        BBSS    #11,FP,.+1
580$:   BBC     #7,R1,590$
        BBSS    #24,R2,.+1
        BBSS    #27,R4,.+1
        BBSS    #3,R6,.+1
        BBSS    #21,R8,.+1
        BBSS    #2,R9,.+1
        BBSS    #30,R11,.+1
        BBSS    #7,R12,.+1
590$:   BBC     #6,R1,600$
        BBSS    #14,R2,.+1
        BBSS    #28,R4,.+1
        BBSS    #26,R5,.+1
        BBSS    #9,R6,.+1
        BBSS    #27,R8,.+1
        BBSS    #27,R10,.+1
        BBSS    #3,R12,.+1
600$:   BBC     #5,R1,610$
        BBSS    #13,R2,.+1
        BBSS    #7,R3,.+1
        BBSS    #9,R5,.+1
        BBSS    #13,R6,.+1
        BBSS    #28,R10,.+1
        BBSS    #26,R11,.+1
        BBSS    #9,R12,.+1
610$:   BBC     #4,R1,620$
        BBSS    #28,R2,.+1
        BBSS    #3,R3,.+1
```

```
        BBSS    #21,R5,.+1
        BBSS    #2,R6,.+1
        BBSS    #30,R8,.+1
        BBSS    #7,R9,.+1
        BBSS    #9,R11,.+1
        BBSS    #13,R12,.+1
620$:   BBC     #3,R1,630$
        BBSS    #27,R3,.+1
        BBSS    #7,R5,.+1
        BBSS    #9,R7,.+1
        BBSS    #4,R8,.+1
        BBSS    #29,R12,.+1
        BBSS    #21,FP,.+1
630$:   BBC     #2,R1,640$
        BBSS    #18,R3,.+1
        BBSS    #29,R6,.+1
        BBSS    #21,R7,.+1
        BBSS    #17,R9,.+1
        BBSS    #18,R10,.+1
        BBSS    #22,R12,.+1
        BBSS    #15,FP,.+1
640$:   BBC     #1,R1,650$
        BBSS    #24,R3,.+1
        BBSS    #18,R4,.+1
        BBSS    #22,R6,.+1
        BBSS    #15,R7,.+1
        BBSS    #26,R9,.+1
        BBSS    #19,R10,.+1
        BBSS    #30,R12,.+1
;
650$:   MOVW    R3,K_EYS+10
        ROTL    #16,R3,R3
        MOVW    R3,K_EYS+0
        MOVL    R2,K_EYS+2
        MOVL    R4,K_EYS+6
;
        MOVW    R6,K_EYS+22
        ROTL    #16,R6,R6
        MOVW    R6,K_EYS+12
        MOVL    R5,K_EYS+14
        MOVL    R7,K_EYS+18
;
        MOVW    R9,K_EYS+34
        ROTL    #16,R9,R9
        MOVW    R9,K_EYS+24
        MOVL    R8,K_EYS+26
        MOVL    R10,K_EYS+30
;
        MOVW    R12,K_EYS+46
        ROTL    #16,R12,R12
        MOVW    R12,K_EYS+36
        MOVL    R11,K_EYS+38
        MOVL    FP,K_EYS+42
;
        CLRQ    R2
        CLRQ    R4
        CLRQ    R6
```

```
        CLRQ    R8
        CLRQ    R10
        CLRQ    R12
;
        BBC     #31,R0,660$
        BBSS    #21,R2,.+1
        BBSS    #2,R3,.+1
        BBSS    #30,R5,.+1
        BBSS    #7,R6,.+1
        BBSS    #9,R8,.+1
        BBSS    #13,R9,.+1
        BBSS    #30,FP,.+1
660$:   BBC     #30,R0,670$
        BBSS    #27,R2,.+1
        BBSS    #27,R4,.+1
        BBSS    #3,R6,.+1
        BBSS    #21,R8,.+1
        BBSS    #2,R9,.+1
        BBSS    #30,R11,.+1
        BBSS    #29,FP,.+1
670$:   BBC     #29,R0,680$
        BBSS    #28,R4,.+1
        BBSS    #26,R5,.+1
        BBSS    #9,R6,.+1
        BBSS    #27,R8,.+1
        BBSS    #27,R10,.+1
        BBSS    #12,R12,.+1
680$:   BBC     #28,R0,690$
        BBSS    #7,R2,.+1
        BBSS    #9,R4,.+1
        BBSS    #4,R5,.+1
        BBSS    #29,R9,.+1
        BBSS    #21,R10,.+1
        BBSS    #17,R12,.+1
        BBSS    #8,FP,.+1
690$:   BBC     #27,R0,700$
        BBSS    #4,R2,.+1
        BBSS    #29,R6,.+1
        BBSS    #21,R7,.+1
        BBSS    #17,R9,.+1
        BBSS    #18,R10,.+1
        BBSS    #22,R12,.+1
        BBSS    #22,FP,.+1
700$:   BBC     #26,R0,710$
        BBSS    #17,R3,.+1
        BBSS    #18,R4,.+1
        BBSS    #22,R6,.+1
        BBSS    #15,R7,.+1
        BBSS    #26,R9,.+1
        BBSS    #19,R10,.+1
        BBSS    #30,R12,.+1
        BBSS    #11,FP,.+1
710$:   BBC     #25,R0,720$
        BBSS    #26,R3,.+1
        BBSS    #19,R4,.+1
        BBSS    #30,R6,.+1
        BBSS    #7,R8,.+1
```

```
        BBSS    #9,R10,.+1
        BBSS    #4,R11,.+1
        BBSS    #2,FP,.+1
720$:   BBC     #23,R0,730$
        BBSS    #10,R2,.+1
        BBSS    #13,R5,.+1
        BBSS    #15,R6,.+1
        BBSS    #17,R8,.+1
        BBSS    #4,R9,.+1
        BBSS    #14,R11,.+1
        BBSS    #28,FP,.+1
730$:   BBC     #22,R0,740$
        BBSS    #24,R2,.+1
        BBSS    #31,R4,.+1
        BBSS    #28,R5,.+1
        BBSS    #10,R8,.+1
        BBSS    #13,R11,.+1
        BBSS    #7,R12,.+1
740$:   BBC     #21,R0,750$
        BBSS    #14,R2,.+1
        BBSS    #6,R3,.+1
        BBSS    #16,R5,.+1
        BBSS    #24,R7,.+1
        BBSS    #24,R8,.+1
        BBSS    #31,R10,.+1
        BBSS    #28,R11,.+1
        BBSS    #3,R12,.+1
750$:   BBC     #20,R0,760$
        BBSS    #0,R2,.+1
        BBSS    #18,R6,.+1
        BBSS    #7,R7,.+1
        BBSS    #16,R9,.+1
        BBSS    #17,R10,.+1
        BBSS    #24,R12,.+1
        BBSS    #18,FP,.+1
760$:   BBC     #19,R0,770$
        BBSS    #18,R3,.+1
        BBSS    #7,R4,.+1
        BBSS    #16,R6,.+1
        BBSS    #17,R7,.+1
        BBSS    #24,R9,.+1
        BBSS    #6,R11,.+1
        BBSS    #15,FP,.+1
770$:   BBC     #18,R0,780$
        BBSS    #24,R3,.+1
        BBSS    #6,R5,.+1
        BBSS    #3,R7,.+1
        BBSS    #28,R9,.+1
        BBSS    #5,R10,.+1
        BBSS    #27,R12,.+1
780$:   BBC     #17,R0,790$
        BBSS    #28,R3,.+1
        BBSS    #5,R4,.+1
        BBSS    #27,R6,.+1
        BBSS    #4,R7,.+1
        BBSS    #0,R8,.+1
        BBSS    #18,R12,.+1
```

```
790$:   BBC     #15,R0,800$
        BBSS    #18,R2,.+1
        BBSS    #14,R3,.+1
        BBSS    #23,R5,.+1
        BBSS    #25,R7,.+1
        BBSS    #29,R8,.+1
        BBSS    #12,R11,.+1
        BBSS    #6,R12,.+1
800$:   BBC     #14,R0,810$
        BBSS    #11,R2,.+1
        BBSS    #19,R5,.+1
        BBSS    #5,R6,.+1
        BBSS    #18,R8,.+1
        BBSS    #14,R9,.+1
        BBSS    #23,R11,.+1
        BBSS    #15,R12,.+1
810$:   BBC     #13,R0,820$
        BBSS    #12,R2,.+1
        BBSS    #10,R3,.+1
        BBSS    #25,R5,.+1
        BBSS    #11,R6,.+1
        BBSS    #11,R8,.+1
        BBSS    #19,R11,.+1
820$:   BBC     #12,R0,830$
        BBSS    #25,R3,.+1
        BBSS    #20,R4,.+1
        BBSS    #13,R7,.+1
        BBSS    #5,R8,.+1
        BBSS    #1,R10,.+1
        BBSS    #2,R11,.+1
830$:   BBC     #11,R0,840$
        BBSS    #13,R4,.+1
        BBSS    #5,R5,.+1
        BBSS    #1,R7,.+1
        BBSS    #2,R8,.+1
        BBSS    #6,R10,.+1
        BBSS    #31,R12,.+1
        BBSS    #3,FP,.+1
840$:   BBC     #10,R0,850$
        BBSS    #2,R2,.+1
        BBSS    #6,R4,.+1
        BBSS    #31,R6,.+1
        BBSS    #10,R7,.+1
        BBSS    #3,R8,.+1
        BBSS    #14,R10,.+1
        BBSS    #4,FP,.+1
850$:   BBC     #9,R0,860$
        BBSS    #3,R2,.+1
        BBSS    #14,R4,.+1
        BBSS    #23,R7,.+1
        BBSS    #25,R9,.+1
        BBSS    #20,R10,.+1
        BBSS    #7,FP,.+1
860$:   BBC     #7,R0,870$
        BBSS    #29,R4,.+1
        BBSS    #31,R5,.+1
        BBSS    #1,R6,.+1
```

```
        BBSS    #20,R8,.+1
        BBSS    #30,R10,.+1
        BBSS    #22,R11,.+1
        BBSS    #10,R12,.+1
870$:   BBC     #6,R0,880$
        BBSS    #15,R2,.+1
        BBSS    #12,R3,.+1
        BBSS    #26,R7,.+1
        BBSS    #29,R10,.+1
        BBSS    #31,R11,.+1
        BBSS    #25,FP,.+1
880$:   BBC     #5,R0,890$
        BBSS    #22,R2,.+1
        BBSS    #0,R3,.+1
        BBSS    #8,R5,.+1
        BBSS    #8,R6,.+1
        BBSS    #15,R8,.+1
        BBSS    #12,R9,.+1
        BBSS    #5,R12,.+1
890$:   BBC     #4,R0,900$
        BBSS    #2,R4,.+1
        BBSS    #23,R6,.+1
        BBSS    #0,R7,.+1
        BBSS    #1,R8,.+1
        BBSS    #8,R10,.+1
        BBSS    #6,FP,.+1
900$:   BBC     #3,R0,910$
        BBSS    #23,R3,.+1
        BBSS    #0,R4,.+1
        BBSS    #1,R5,.+1
        BBSS    #8,R7,.+1
        BBSS    #22,R10,.+1
        BBSS    #19,R12,.+1
        BBSS    #10,FP,.+1
910$:   BBC     #2,R0,920$
        BBSS    #22,R4,.+1
        BBSS    #19,R6,.+1
        BBSS    #12,R7,.+1
        BBSS    #21,R9,.+1
        BBSS    #11,R10,.+1
        BBSS    #20,R12,.+1
        BBSS    #23,FP,.+1
920$:   BBC     #1,R0,930$
        BBSS    #21,R3,.+1
        BBSS    #11,R4,.+1
        BBSS    #20,R6,.+1
        BBSS    #16,R7,.+1
        BBSS    #2,R10,.+1
        BBSS    #23,R12,.+1
        BBSS    #13,FP,.+1
930$:   BBC     #31,R1,940$
        BBSS    #30,R2,.+1
        BBSS    #7,R3,.+1
        BBSS    #9,R5,.+1
        BBSS    #13,R6,.+1
        BBSS    #28,R10,.+1
        BBSS    #26,R11,.+1
```

```
        BBSS    #0,R12,.+1
940$:   BBC     #30,R1,950$
        BBSS    #3,R3,.+1
        BBSS    #21,R5,.+1
        BBSS    #2,R6,.+1
        BBSS    #30,R8,.+1
        BBSS    #7,R9,.+1
        BBSS    #9,R11,.+1
        BBSS    #1,R12,.+1
950$:   BBC     #29,R1,960$
        BBSS    #26,R2,.+1
        BBSS    #9,R3,.+1
        BBSS    #27,R5,.+1
        BBSS    #27,R7,.+1
        BBSS    #3,R9,.+1
        BBSS    #21,R11,.+1
        BBSS    #26,FP,.+1
960$:   BBC     #28,R1,970$
        BBSS    #9,R2,.+1
        BBSS    #13,R3,.+1
        BBSS    #28,R7,.+1
        BBSS    #26,R8,.+1
        BBSS    #9,R9,.+1
        BBSS    #27,R11,.+1
        BBSS    #8,R12,.+1
970$:   BBC     #27,R1,980$
        BBSS    #29,R3,.+1
        BBSS    #21,R4,.+1
        BBSS    #17,R6,.+1
        BBSS    #18,R7,.+1
        BBSS    #22,R9,.+1
        BBSS    #15,R10,.+1
        BBSS    #26,R12,.+1
        BBSS    #12,FP,.+1
980$:   BBC     #26,R1,990$
        BBSS    #22,R3,.+1
        BBSS    #15,R4,.+1
        BBSS    #26,R6,.+1
        BBSS    #19,R7,.+1
        BBSS    #30,R9,.+1
        BBSS    #7,R11,.+1
        BBSS    #16,FP,.+1
990$:   BBC     #25,R1,1000$
        BBSS    #30,R3,.+1
        BBSS    #7,R5,.+1
        BBSS    #9,R7,.+1
        BBSS    #4,R8,.+1
        BBSS    #29,R12,.+1
        BBSS    #0,FP,.+1
1000$:  BBC     #23,R1,1010$
        BBSS    #13,R2,.+1
        BBSS    #15,R3,.+1
        BBSS    #17,R5,.+1
        BBSS    #4,R6,.+1
        BBSS    #14,R8,.+1
        BBSS    #6,R9,.+1
        BBSS    #16,R11,.+1
```

```
        BBSS    #9,R12,.+1
1010$:  BBC     #22,R1,1020$
        BBSS    #28,R2,.+1
        BBSS    #10,R5,.+1
        BBSS    #13,R8,.+1
        BBSS    #15,R9,.+1
        BBSS    #17,R11,.+1
        BBSS    #13,R12,.+1
1020$:  BBC     #21,R1,1030$
        BBSS    #16,R2,.+1
        BBSS    #24,R4,.+1
        BBSS    #24,R5,.+1
        BBSS    #31,R7,.+1
        BBSS    #28,R8,.+1
        BBSS    #10,R11,.+1
        BBSS    #2,R12,.+1
1030$:  BBC     #20,R1,1040$
        BBSS    #17,R2,.+1
        BBSS    #4,R3,.+1
        BBSS    #14,R5,.+1
        BBSS    #6,R6,.+1
        BBSS    #16,R8,.+1
        BBSS    #24,R10,.+1
        BBSS    #24,R11,.+1
        BBSS    #27,FP,.+1
1040$:  BBC     #19,R1,1050$
        BBSS    #16,R3,.+1
        BBSS    #17,R4,.+1
        BBSS    #24,R6,.+1
        BBSS    #6,R8,.+1
        BBSS    #3,R10,.+1
        BBSS    #28,R12,.+1
        BBSS    #19,FP,.+1
1050$:  BBC     #18,R1,1060$
        BBSS    #6,R2,.+1
        BBSS    #3,R4,.+1
        BBSS    #28,R6,.+1
        BBSS    #5,R7,.+1
        BBSS    #27,R9,.+1
        BBSS    #4,R10,.+1
        BBSS    #0,R11,.+1
        BBSS    #9,FP,.+1
1060$:  BBC     #17,R1,1070$
        BBSS    #27,R3,.+1
        BBSS    #4,R4,.+1
        BBSS    #0,R5,.+1
        BBSS    #18,R9,.+1
        BBSS    #7,R10,.+1
        BBSS    #16,R12,.+1
        BBSS    #21,FP,.+1
1070$:  BBC     #15,R1,1080$
        BBSS    #23,R2,.+1
        BBSS    #25,R4,.+1
        BBSS    #29,R5,.+1
        BBSS    #12,R8,.+1
        BBSS    #10,R9,.+1
        BBSS    #25,R11,.+1
```

```
        BBSS    #24,FP,.+1
1080$:  BBC     #14,R1,1090$
        BBSS    #19,R2,.+1
        BBSS    #5,R3,.+1
        BBSS    #18,R5,.+1
        BBSS    #14,R6,.+1
        BBSS    #23,R8,.+1
        BBSS    #25,R10,.+1
        BBSS    #29,R11,.+1
        BBSS    #4,R12,.+1
1090$:  BBC     #13,R1,1100$
        BBSS    #25,R2,.+1
        BBSS    #11,R3,.+1
        BBSS    #11,R5,.+1
        BBSS    #19,R8,.+1
        BBSS    #5,R9,.+1
        BBSS    #18,R11,.+1
1100$:  BBC     #12,R1,1110$
        BBSS    #29,R2,.+1
        BBSS    #12,R5,.+1
        BBSS    #10,R6,.+1
        BBSS    #25,R8,.+1
        BBSS    #11,R9,.+1
        BBSS    #11,R11,.+1
        BBSS    #31,FP,.+1
1110$:  BBC     #11,R1,1120$
        BBSS    #5,R2,.+1
        BBSS    #1,R4,.+1
        BBSS    #2,R5,.+1
        BBSS    #6,R7,.+1
        BBSS    #31,R9,.+1
        BBSS    #10,R10,.+1
        BBSS    #3,R11,.+1
        BBSS    #5,FP,.+1
1120$:  BBC     #10,R1,1130$
        BBSS    #31,R3,.+1
        BBSS    #10,R4,.+1
        BBSS    #3,R5,.+1
        BBSS    #14,R7,.+1
        BBSS    #23,R10,.+1
        BBSS    #25,R12,.+1
1130$:  BBC     #9,R1,1140$
        BBSS    #23,R4,.+1
        BBSS    #25,R6,.+1
        BBSS    #20,R7,.+1
        BBSS    #13,R10,.+1
        BBSS    #5,R11,.+1
        BBSS    #17,FP,.+1
1140$:  BBC     #7,R1,1150$
        BBSS    #31,R2,.+1
        BBSS    #1,R3,.+1
        BBSS    #20,R5,.+1
        BBSS    #30,R7,.+1
        BBSS    #22,R8,.+1
        BBSS    #0,R9,.+1
        BBSS    #8,R11,.+1
        BBSS    #11,R12,.+1
```

```
1150$:  BBC     #6,R1,1160$
        BBSS    #26,R4,.+1
        BBSS    #29,R7,.+1
        BBSS    #31,R8,.+1
        BBSS    #1,R9,.+1
        BBSS    #20,R11,.+1
1160$:  BBC     #5,R1,1170$
        BBSS    #8,R2,.+1
        BBSS    #8,R3,.+1
        BBSS    #15,R5,.+1
        BBSS    #12,R6,.+1
        BBSS    #26,R10,.+1
        BBSS    #14,R12,.+1
1170$:  BBC     #4,R1,1180$
        BBSS    #20,R2,.+1
        BBSS    #30,R4,.+1
        BBSS    #22,R5,.+1
        BBSS    #0,R6,.+1
        BBSS    #8,R8,.+1
        BBSS    #8,R9,.+1
        BBSS    #15,R11,.+1
1180$:  BBC     #3,R1,1190$
        BBSS    #1,R2,.+1
        BBSS    #8,R4,.+1
        BBSS    #22,R7,.+1
        BBSS    #19,R9,.+1
        BBSS    #12,R10,.+1
        BBSS    #21,R12,.+1
        BBSS    #14,FP,.+1
1190$:  BBC     #2,R1,1200$
        BBSS    #19,R3,.+1
        BBSS    #12,R4,.+1
        BBSS    #21,R6,.+1
        BBSS    #11,R7,.+1
        BBSS    #20,R9,.+1
        BBSS    #16,R10,.+1
        BBSS    #20,FP,.+1
1200$:  BBC     #1,R1,1210$
        BBSS    #20,R3,.+1
        BBSS    #16,R4,.+1
        BBSS    #2,R7,.+1
        BBSS    #23,R9,.+1
        BBSS    #0,R10,.+1
        BBSS    #1,R11,.+1
        BBSS    #1,FP,.+1
;
1210$:  MOVW    R3,K_EYS+58
        ROTL    #16,R3,R3
        MOVW    R3,K_EYS+48
        MOVL    R2,K_EYS+50
        MOVL    R4,K_EYS+54
;
        MOVW    R6,K_EYS+70
        ROTL    #16,R6,R6
        MOVW    R6,K_EYS+60
        MOVL    R5,K_EYS+62
        MOVL    R7,K_EYS+66
```

```
;
        MOVW    R9,K_EYS+82
        ROTL    #16,R9,R9
        MOVW    R9,K_EYS+72
        MOVL    R8,K_EYS+74
        MOVL    R10,K_EYS+78
;
        MOVW    R12,K_EYS+94
        ROTL    #16,R12,R12
        MOVW    R12,K_EYS+84
        MOVL    R11,K_EYS+86
        MOVL    FP,K_EYS+90
;
        MOVQ    SREG,R12                ; RESTORE REGISTERS R12 & R13
;
        RET
;
;       ************************************************
;       *                                              *
;       *               ENTRY POINT SET_MODE           *
;       *                                              *
;       ************************************************
;
; CALLING SEQUENCE:
;
;                 CALL SET_MODE (MODE)
;
; PURPOSE:
;
;       SET MODE OF ENCRYPTION TO CBC OR ECB.
;
; ARGUMENTS:
;
;       MODE    :  MODE FLAG:
;
;                       1 = CYPHER BLOCK CHAINING MODE
;                       2 = ELECTRONIC CODE BOOK MODE
;
;
; OFFSETS FOR ARGUMENT LIST
;
        MODE=4
;
        .ENTRY SET_MODE,0
;
        MOVL    @MODE(AP),M_ODE
        RET
;
;
;       ************************************************
;       *                                              *
;       *               ENTRY POINT SET_DIR            *
;       *                                              *
;       ************************************************
;
; CALLING SEQUENCE:
;
```

```
;                   CALL SET_DIR (DIR)
;
; PURPOSE:
;
;       SET DIRECTION TO ENCRYPT OR DECRYPT.
;
; ARGUMENTS:
;
;       DIR     :  DIRECTION FLAG:
;
;                       1 = SET DIRECTION TO ENCRYPTION.
;                       2 = SET DIRECTION TO DECRYPTION.
;
;
; OFFSETS FOR ARGUMENT LIST
;
        DIR=4
;
        .ENTRY SET_DIR,0
;
        MOVL    @DIR(AP),D_IR
        RET
;
;       ************************************************
;       *                                              *
;       *               ENTRY POINT PDES               *
;       *                                              *
;       ************************************************
;
; CALLING SEQUENCE:
;
;                   CALL PDES (INSTR,OUTSTR,LEN,IVEC)
;
; PURPOSE:
;
;       PERFORM NBS ENCRYPTION/DECRYPTION OF INPUT STRING.
;
; ARGUMENTS:
;
;       INSTR   :  INPUT STRING.
;
;       OUTSTR  := OUTPUT STRING.
;
;       LEN     :  LENGTH OF STRING IN 64-BIT QUADWORDS.
;
;       IVEC    := FINAL VALUE OF VECTOR
;
;
; OFFSETS FOR ARGUMENT LIST
;
        INSTR=4
        OUTSTR=8
        LEN=12
        IVEC=16
;
        .ENTRY PDES ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
;
```

```
;
        MOVL    INSTR(AP),I_ADD
        MOVL    OUTSTR(AP),O_ADD
        MOVL    @LEN(AP),L_EN
;
; MAIN LOOP OVER QUADWORDS
;
LOOPA:  MOVQ    @I_ADD,R2               ; R0,R1 = INPT
        MOVL    D_IR,R11                ; R11 = DIRECTION
;
; CHECK FOR CYPHER BLOCK CHAINING
;
        BBC     #0,M_ODE,2100$
        BBC     #0,R11,2100$
        XORL2   I_VEC,R2                ; CBC ENCRYPTION
        XORL2   I_VEC+4,R3
;

2100$:  MOVL    R2,R1
        MOVL    R3,R0
;
; APPLY INITIAL PERMUTATIONS IP AND E.
;
        CLRQ    R2                      ; R2,R3 WILL CONTAIN E(IP(R))
        CLRQ    R4                      ; R4,R5 WILL CONTAIN E(IP(L))
;
        BBC     #00,R0,.+9
        BBSS    #06,R2,.+1
        BBSS    #04,R2,.+1
        BBC     #01,R0,.+9
        BBSS    #06,R4,.+1
        BBSS    #04,R4,.+1
        BBC     #02,R0,.+9
        BBSS    #18,R2,.+1
        BBSS    #16,R2,.+1
        BBC     #03,R0,.+9
        BBSS    #18,R4,.+1
        BBSS    #16,R4,.+1
        BBC     #04,R0,.+9
        BBSS    #28,R2,.+1
        BBSS    #30,R2,.+1
        BBC     #05,R0,.+9
        BBSS    #28,R4,.+1
        BBSS    #30,R4,.+1
        BBC     #06,R0,.+9
        BBSS    #08,R3,.+1
        BBSS    #10,R3,.+1
        BBC     #07,R0,.+9
        BBSS    #08,R5,.+1
        BBSS    #10,R5,.+1
        BBC     #08,R0,.+5
        BBSS    #03,R2,.+1
        BBC     #09,R0,.+5
        BBSS    #03,R4,.+1
        BBC     #10,R0,.+5
        BBSS    #15,R2,.+1
        BBC     #11,R0,.+5
```

```
BBSS    #15,R4,.+1
BBC     #12,R0,.+5
BBSS    #27,R2,.+1
BBC     #13,R0,.+5
BBSS    #27,R4,.+1
BBC     #14,R0,.+5
BBSS    #07,R3,.+1
BBC     #15,R0,.+5
BBSS    #07,R5,.+1
BBC     #16,R0,.+5
BBSS    #02,R2,.+1
BBC     #17,R0,.+5
BBSS    #02,R4,.+1
BBC     #18,R0,.+5
BBSS    #14,R2,.+1
BBC     #19,R0,.+5
BBSS    #14,R4,.+1
BBC     #20,R0,.+5
BBSS    #26,R2,.+1
BBC     #21,R0,.+5
BBSS    #26,R4,.+1
BBC     #22,R0,.+5
BBSS    #06,R3,.+1
BBC     #23,R0,.+5
BBSS    #06,R5,.+1
BBC     #24,R0,.+9
BBSS    #15,R3,.+1
BBSS    #01,R2,.+1
BBC     #25,R0,.+9
BBSS    #15,R5,.+1
BBSS    #01,R4,.+1
BBC     #26,R0,.+9
BBSS    #13,R2,.+1
BBSS    #11,R2,.+1
BBC     #27,R0,.+9
BBSS    #13,R4,.+1
BBSS    #11,R4,.+1
BBC     #28,R0,.+9
BBSS    #25,R2,.+1
BBSS    #23,R2,.+1
BBC     #29,R0,.+9
BBSS    #25,R4,.+1
BBSS    #23,R4,.+1
BBC     #30,R0,.+9
BBSS    #05,R3,.+1
BBSS    #03,R3,.+1
BBC     #31,R0,.+9
BBSS    #05,R5,.+1
BBSS    #03,R5,.+1
BBC     #00,R1,.+9
BBSS    #12,R2,.+1
BBSS    #10,R2,.+1
BBC     #01,R1,.+9
BBSS    #12,R4,.+1
BBSS    #10,R4,.+1
BBC     #02,R1,.+9
BBSS    #24,R2,.+1
```

```
BBSS    #22,R2,.+1
BBC     #03,R1,.+9
BBSS    #24,R4,.+1
BBSS    #22,R4,.+1
BBC     #04,R1,.+9
BBSS    #04,R3,.+1
BBSS    #02,R3,.+1
BBC     #05,R1,.+9
BBSS    #04,R5,.+1
BBSS    #02,R5,.+1
BBC     #06,R1,.+9
BBSS    #14,R3,.+1
BBSS    #00,R2,.+1
BBC     #07,R1,.+9
BBSS    #14,R5,.+1
BBSS    #00,R4,.+1
BBC     #08,R1,.+5
BBSS    #09,R2,.+1
BBC     #09,R1,.+5
BBSS    #09,R4,.+1
BBC     #10,R1,.+5
BBSS    #21,R2,.+1
BBC     #11,R1,.+5
BBSS    #21,R4,.+1
BBC     #12,R1,.+5
BBSS    #01,R3,.+1
BBC     #13,R1,.+5
BBSS    #01,R5,.+1
BBC     #14,R1,.+5
BBSS    #13,R3,.+1
BBC     #15,R1,.+5
BBSS    #13,R5,.+1
BBC     #16,R1,.+5
BBSS    #08,R2,.+1
BBC     #17,R1,.+5
BBSS    #08,R4,.+1
BBC     #18,R1,.+5
BBSS    #20,R2,.+1
BBC     #19,R1,.+5
BBSS    #20,R4,.+1
BBC     #20,R1,.+5
BBSS    #00,R3,.+1
BBC     #21,R1,.+5
BBSS    #00,R5,.+1
BBC     #22,R1,.+5
BBSS    #12,R3,.+1
BBC     #23,R1,.+5
BBSS    #12,R5,.+1
BBC     #24,R1,.+9
BBSS    #07,R2,.+1
BBSS    #05,R2,.+1
BBC     #25,R1,.+9
BBSS    #07,R4,.+1
BBSS    #05,R4,.+1
BBC     #26,R1,.+9
BBSS    #19,R2,.+1
BBSS    #17,R2,.+1
```

```
        BBC     #27,R1,.+9
        BBSS    #19,R4,.+1
        BBSS    #17,R4,.+1
        BBC     #28,R1,.+9
        BBSS    #29,R2,.+1
        BBSS    #31,R2,.+1
        BBC     #29,R1,.+9
        BBSS    #29,R4,.+1
        BBSS    #31,R4,.+1
        BBC     #30,R1,.+9
        BBSS    #09,R3,.+1
        BBSS    #11,R3,.+1
        BBC     #31,R1,.+9
        BBSS    #09,R5,.+1
        BBSS    #11,R5,.+1
;
; ASSUME ENCRYPTION
;
        CLRL    R9
        MOVL    #6,R10
        BBS     #0,R11,2000$            ; ENCRYPTION
;
        MOVL    #90,R9
        MNEGL   R10,R10                 ; DECRYPTION
;
2000$:  MOVL    #1,R11                  ; ITER = 1
;
; TOP OF MAIN LOOP. REGISTER USAGE IS AS FOLLOWS:
;
;               R0  = K(I) .XOR. E(R(I))
;               R1  = K(I) .XOR. E(R(I))
;               R2  = E(L)
;               R3  = E(L)
;               R4  = E(R)
;               R5  = E(R)
;               R6  = WORK AREA
;               R7  = WORK AREA
;               R8  = INDEX TO S-BOXES
;               R9  = OFFSET TO NEXT KEY
;               R10 = KEY ADDRESS INCREMENT
;               R11 = ITERATION COUNTER
LOOP:
;
        XORW3   L^K_EYS+4(R9),R5,R1
        XORL3   L^K_EYS(R9),R4,R0
        CLRQ    R6
;
        EXTZV   #0,#6,R0,R8
        BISW2   S_8A[R8],R7                     ; OR S_8 INTO RESULT
        BISL2   S_8B[R8],R6                     ;
;
        EXTZV   #6,#6,R0,R8
        BISW2   S_7A[R8],R7                     ; OR S_7 INTO RESULT
        BISL2   S_7B[R8],R6                     ;
;
        EXTZV   #12,#6,R0,R8
        BISW2   S_6A[R8],R7                     ; OR S_6 INTO RESULT
```

```
        BISL2   S_6B[R8],R6                     ;
;
        EXTZV   #18,#6,R0,R8
        BISW2   S_5A[R8],R7                     ; OR S_5 INTO RESULT
        BISL2   S_5B[R8],R6                     ;
;
        EXTZV   #24,#6,R0,R8
        BISW2   S_4A[R8],R7                     ; OR S_4 INTO RESULT
        BISL2   S_4B[R8],R6                     ;
;
        ASHQ    #2,R0,R0
;
        EXTZV   #0,#6,R1,R8
        BISW2   S_3A[R8],R7                     ; OR S_3 INTO RESULT
        BISL2   S_3B[R8],R6                     ;
;
        EXTZV   #6,#6,R1,R8
        BISW2   S_2A[R8],R7                     ; OR S_2 INTO RESULT
        BISL2   S_2B[R8],R6                     ;
;
        EXTZV   #12,#6,R1,R8
        BISW2   S_1A[R8],R7                     ; OR S_1 INTO RESULT
        BISL2   S_1B[R8],R6                     ;
;
        XORL2   R2,R6
        XORL2   R3,R7
        MOVQ    R4,R2                   ; E(L') = E(R)
        MOVQ    R6,R4                   ; E(R') = E(L).XOR.F(E(R),K)
;
; INCREMENT COUNTERS
;
        ADDL2   R10,R9                  ; OFFSET TO KEYS
        ACBW    #16,#1,R11,LOOP         ; 16 ITERATIONS
;
; PERFORM INVERSES OF PERMUTATIONS E AND IP.
;
        CLRQ    R0                      ; PERMUTE R2,R3,R4,R5 TO R0,R1
;
        BBC     #00,R5,.+5
        BBSS    #20,R0,.+1
        BBC     #01,R5,.+5
        BBSS    #12,R0,.+1
        BBC     #04,R5,.+5
        BBSS    #04,R0,.+1
        BBC     #05,R5,.+5
        BBSS    #30,R1,.+1
        BBC     #06,R5,.+5
        BBSS    #22,R1,.+1
        BBC     #07,R5,.+5
        BBSS    #14,R1,.+1
        BBC     #10,R5,.+5
        BBSS    #06,R1,.+1
        BBC     #11,R5,.+5
        BBSS    #30,R0,.+1
        BBC     #12,R5,.+5
        BBSS    #22,R0,.+1
        BBC     #13,R5,.+5
```

```
BBSS    #14,R0,.+1
BBC     #14,R5,.+5
BBSS    #06,R0,.+1
BBC     #15,R5,.+5
BBSS    #24,R1,.+1
BBC     #02,R4,.+5
BBSS    #16,R1,.+1
BBC     #03,R4,.+5
BBSS    #08,R1,.+1
BBC     #06,R4,.+5
BBSS    #00,R1,.+1
BBC     #07,R4,.+5
BBSS    #24,R0,.+1
BBC     #08,R4,.+5
BBSS    #16,R0,.+1
BBC     #09,R4,.+5
BBSS    #08,R0,.+1
BBC     #12,R4,.+5
BBSS    #00,R0,.+1
BBC     #13,R4,.+5
BBSS    #26,R1,.+1
BBC     #14,R4,.+5
BBSS    #18,R1,.+1
BBC     #15,R4,.+5
BBSS    #10,R1,.+1
BBC     #18,R4,.+5
BBSS    #02,R1,.+1
BBC     #19,R4,.+5
BBSS    #26,R0,.+1
BBC     #20,R4,.+5
BBSS    #18,R0,.+1
BBC     #21,R4,.+5
BBSS    #10,R0,.+1
BBC     #24,R4,.+5
BBSS    #02,R0,.+1
BBC     #25,R4,.+5
BBSS    #28,R1,.+1
BBC     #26,R4,.+5
BBSS    #20,R1,.+1
BBC     #27,R4,.+5
BBSS    #12,R1,.+1
BBC     #30,R4,.+5
BBSS    #04,R1,.+1
BBC     #31,R4,.+5
BBSS    #28,R0,.+1
BBC     #00,R3,.+5
BBSS    #21,R0,.+1
BBC     #01,R3,.+5
BBSS    #13,R0,.+1
BBC     #04,R3,.+5
BBSS    #05,R0,.+1
BBC     #05,R3,.+5
BBSS    #31,R1,.+1
BBC     #06,R3,.+5
BBSS    #23,R1,.+1
BBC     #07,R3,.+5
BBSS    #15,R1,.+1
```

```
        BBC     #10,R3,.+5
        BBSS    #07,R1,.+1
        BBC     #11,R3,.+5
        BBSS    #31,R0,.+1
        BBC     #12,R3,.+5
        BBSS    #23,R0,.+1
        BBC     #13,R3,.+5
        BBSS    #15,R0,.+1
        BBC     #14,R3,.+5
        BBSS    #07,R0,.+1
        BBC     #15,R3,.+5
        BBSS    #25,R1,.+1
        BBC     #02,R2,.+5
        BBSS    #17,R1,.+1
        BBC     #03,R2,.+5
        BBSS    #09,R1,.+1
        BBC     #06,R2,.+5
        BBSS    #01,R1,.+1
        BBC     #07,R2,.+5
        BBSS    #25,R0,.+1
        BBC     #08,R2,.+5
        BBSS    #17,R0,.+1
        BBC     #09,R2,.+5
        BBSS    #09,R0,.+1
        BBC     #12,R2,.+5
        BBSS    #01,R0,.+1
        BBC     #13,R2,.+5
        BBSS    #27,R1,.+1
        BBC     #14,R2,.+5
        BBSS    #19,R1,.+1
        BBC     #15,R2,.+5
        BBSS    #11,R1,.+1
        BBC     #18,R2,.+5
        BBSS    #03,R1,.+1
        BBC     #19,R2,.+5
        BBSS    #27,R0,.+1
        BBC     #20,R2,.+5
        BBSS    #19,R0,.+1
        BBC     #21,R2,.+5
        BBSS    #11,R0,.+1
        BBC     #24,R2,.+5
        BBSS    #03,R0,.+1
        BBC     #25,R2,.+5
        BBSS    #29,R1,.+1
        BBC     #26,R2,.+5
        BBSS    #21,R1,.+1
        BBC     #27,R2,.+5
        BBSS    #13,R1,.+1
        BBC     #30,R2,.+5
        BBSS    #05,R1,.+1
        BBC     #31,R2,.+5
        BBSS    #29,R0,.+1
;
; CHECK FOR CYPHER BLOCK CHAINING
;
        BBC     #0,M_ODE,2300$
        BBC     #0,D_IR,2200$
```

```
        MOVQ    R0,I_VEC                ; CBC ENCRYPTION
        BRB     2300$
;
2200$:  XORL2   I_VEC,R0                ; CBC DECRYPTION
        XORL2   I_VEC+4,R1
        MOVQ    @I_ADD,I_VEC
;
2300$:  MOVQ    R0,@O_ADD               ; STORE RESULT
        ADDL2   #8,O_ADD
        ADDL2   #8,I_ADD
;
        DECL    L_EN
        BEQL    3000$
        BRW     LOOPA
;
; RETURN VECTOR IF CYPHER BLOCK CHAINING
;
3000$:  BBC     #0,M_ODE,9000$
        MOVQ    I_VEC,@IVEC(AP)
;
9000$:  RET
        .END
```

APPENDIX C

USERS GUIDE

I. GENERAL COMMENTS

This users guide provides a walk-through discussion of the actual operation of the WBCN crypto services (WCS). Much of this discussion is repeated in the instructions actually built into the processing modules themselves. This guide must have some starting point to use as a reference. For the purposes of this guide, I have made the assumption that the key management machine (KMM) and the various node gateways are DEC VAX machines that have been fully configured under VMS and have the INGRES DBMS installed and functioning properly on a logical device defined as UD; that the operator has more than a passing understanding of VMS DCL and the INGRES DBMS; and that an account named CRYPTO has been created on all the machines. Further, I assumed that the CRYPTO account is a valid INGRES user account on each machine. That covers the global assumptions for the WCS. Following are the specifics for the three elements of the WCS.

II. THE CENTRALIZED KEY MANAGEMENT

This element of the WCS operates only on the KMM. The code for this element should not be distributed to any of the WBCN nodes. The software for this element is distributed on magnetic tape using the VMS Backup Utility.

The first step is to load the software from the tape to the VMS files. Create a subdirectory [CRYPTO.KEYMGR]. Use the Backup command as follows:

```
$ BACKUP MFAØ: CRYPTO.BCK/SELECT = [CRYPTO.KEYMGR...] [CRYPTO.KEYMGR...]
```

This command transfers all the files and structure from the Backup Save Set CRYPTO.BCK to the VMS directory [CRYPTO.KEYMGR].

Next, set the default directory with the VMS command

```
$ SET DEFAULT [CRYPTO.KEYMGR]
```

Then issue the command

```
$ DIR
```

You should see the screen image shown in Fig. C-1 on the terminal.

Assuming that everything checks out to this point, the next step is to create the database. This is accomplished by running SETUP.COM. You do this with the command

```
$ @ SETUP.COM
```

```
$ dir

Directory UD:[CRYPTO.KEYMGR]

DISTRIB.COM;15     ERROR.COM;3        EXTRACT.COM;15     EXTRACT.EXE;10
INDEX.EXE;13       KEYLOAD.COM;22     LOGICALS.COM;6     NODEPAIR.ING;4
PERIOD.COM;1       ROOT.COM;22        ROOT.COM;21        SETUP.COM;8
SETUP.ING;12       TAPELOAD.ING;10    UPDATE.COM;5

Total of 15 files.
```

Fig. C-1. VMS directory structure resulting from loading the save set CRYPTO.BCK on the KMM.

This process takes a little time so be patient. It is most helpful if you have a separate system manager account active on a separate terminal at this time. If you do, use the MONITOR utility to verify that SETUP.COM is proceeding and not locked up because of some VMS/INGRES setup problem. If SETUP.COM does not complete successfully, get help to determine the problems, fix them, and then run it again.

Upon the successful completion of SETUP.COM, you should be able to verify the existence of the database named KEYMGR, using the CATALOGDB feature of INGRES. Further, you can verify the correct creation of the initial tables by entering the command

```
$ INGRES KEYMGR
```

This starts the basic INGRES process on the database KEYMGR. Enter a help\g <Return> to the INGRES prompt (*). You should see displayed the same screen image as shown in Fig. C-2. Exit the INGRES process by entering \q <Return>. This returns you to the DCL/CLI.

At this point, the KEYMGR database has been created along with the initial tables. There are no data loaded in the database. Before proceeding further, remove SETUP.COM from the system with a

```
$ DELETE/CONF SETUP.COM;1 <Return>
```

All of the remaining centralized key management functions are subordinate to the menu-driven control DCL command procedure ROOT.COM. To proceed with any further activity, start ROOT.COM with the command

```
$ @ ROOT
```

ROOT.COM provides an introduction discussion of its function and eventually displays a menu of selections for the supported activities.

```
$$ ingres     keymgr
INGRES VAX Release 5.0/02a (vax.vms/01) login 13-FEB-1987 11:48:40
Copyright (c) 1986, Relational Technology Inc.

INGRES Release 5.0 (PRODUCTION)
go
* help\g
Executing . . .

name          owner       type        name          owner       type

ptrtable      ingres      table       nodetable     ingres      table
keyindex      ingres      table       prdtable      ingres      table

continue
*
*
*
*
*
*
*
*
```

Fig. C-2. The database structure resulting from running SETUP.COM on the KMM.

III. LOADING KEYS

Loading keys is the next activity that logically follows the creation of the database on the KMM. I assume that the operator has, in hand, an NSA-distributed tape containing the actual key material and needs (wants) to load those keys into the database.

The first thing to do is to make sure there is no write enable ring still in the tape reel. If there is a write ring in the reel, take it out. Next, load the tape on the tape drive. The drive must be capable of 1600-bpi transfer rate because that is how the tape was written. After the tape is loaded and the drive is set to ON-LINE, enter the KEYLOAD option on the ROOT menu selection.

KEYLOAD starts up with a bit of general discussion. It assumes that the device identifier for the tape drive is MTA0:. If such is the case, a <Return> is the correct response to the question. Otherwise enter the correct tape drive device identifier. You should verify the tape drive device identifier before starting KEYLOAD. If you do not know how to do this, ask your system manager.

After you have responded to the tape drive identification question, the process gives you an opportunity to actually load the tape on the drive if you have not already done so. When the tape is loaded, the -READY- response to this question allows the process to continue. Any response other than -READY- forces the process to loop on the inquiry. This process

is set up this way intentionally. The tape distributed by NSA is classified confidential. I decided that keyloading must be run to completion or not started because the tape must be constantly attended. Therefore, there is no convenient mechanism built into the KEYLOAD to cancel the process when it is once started. I recommend it be left that way.

When the tape is loaded and ready, respond -READY- to the inquiry. The remainder of the KEYLOAD process requires no further user interaction. When the tape transfer is complete, KEYLOAD.COM issues a DISMOUNT instruction. This rewinds the tape. After the dismount occurs, the user must remove the tape reel from the drive and return the reel to secured storage. The KEYLOAD.COM procedure issues progress reports to the terminal as it continues loading the keys from the VMS file to the database. When KEYLOAD has completed, it returns to the ROOT.COM menu. This process loads all the keys on the tape. There is no provision for a partial file selection.

IV. THE KEY PERIOD

The key period information management, other than initialization, is automatically taken care of in the WCS code. Initialization is performed using the PERIOD selection from the ROOT.COM menu.

The response to this selection is a Query-By-Forms (QBF) frame that displays two fields: PERIOD and DATA. Use the TAB key to move the cursor from one field to the other. Enter a 1 in the PERIOD field and TODAY in the DATE field. Use the escape key (ESC) to activate the menu selections at the bottom of the screen. Write the data in the table with the GO instruction. Then on the next pass through the query, escape to the menu and exit the process. Continue exits until you return to the ROOT.COM menu. The results of this action establish the beginning of the first key period as the date you first run this process. I would recommend that you not set up the key period until the day you first write the distribution tapes. Otherwise the first period keys may be underutilized.

V. NODE INFORMATION

The next activity prior to key distribution is identifying, in the database, the participating WBCN nodes. This is done by selecting the UPDATE option on the ROOT.COM menu. This selection starts a QBF process that accepts the node identification information. The critical piece of information in the node identification is the NODENAME. This name must be the same as the WBCN node identifier for each WBCN node. This node name must match the response to the DCL command

```
$ SHOW LOGICAL SYSNODE
```

for each WBCN node. The key management and distribution mechanism will reject any distribution at the specific nodes for which these identifiers do not match.

This QBF frame operation is self-guided. Use the TAB key to move from one field to the next and enter the information in all fields. Then

enter escape (ESC) to activate the menu selection. This information can be entered, altered, or updated as necessary. It is not synchronized with any of the other key distribution functions.

There is only one caution. Be sure that the node table is current before starting the key extraction and distribution processes. If the node table is not current for a given distribution, an omitted node will get no keys and the process does not back up.

After you have finished the node identification process, exit from the QBF process back to the main ROOT.COM menu.

VI. KEY SET EXTRACTION

The key set extraction is the first of two processes used to produce the node-specific key distribution tapes. Assuming you have the key period information and node information set up correctly, you invoke the key set extraction process by entering the EXTRACT selection in the main ROOT menu. This process is basically automatic and provides a limited functional description and inquires if the user is ready to continue. If so, the correct response is -READY-. Any other response will force the process to exit and return to the ROOT.COM main menu. This EXTRACT process detects the only really severe problem that the key distribution can have--namely, a shortage of keys. If there are not enough keys in the master table, the error message given in the process ERROR.COM (see code listing) is posted to the terminal. The EXTRACT process will force exit and return to ROOT, which will also force exit. These exits are set up so that the error message remains on the terminal screen when ROOT returns to CLI. If you get this error, the solution is straightforward--load more keys, at least as many as you need for this distribution. The calculation

NNODES*(NNODES - 1)*NBRPERIODS*NKEYS

where

NNODES is the number of WBCN nodes,
NBRPERIODS is the number of key periods, and
NKEYS is the number of keys per period

yields the number of keys needed for a single distribution. The WCS is set to default to

NBRPERIODS = 3 and NKEYS = 20

Thus, the calculation reduces to

60*NNODES*(NNODES - 1)

If you have a problem with the NSA key sources and you need to determine precisely the number of available keys in the master key table, do the following:

(1) Log on to the KMM under the CRYPTO ACCOUNT.
(2) Enter $ INGRES KEYMGR. INGRES returns the prompt (*).
(3) Enter PRINT PTRTABLE\g.

The result of these commands is a display of the pointer table. Read off of that table the PTR values associated with the attributes HIGHPTR and LOWPTR. The difference

PTR(HIGHPTR) - PTR(LOWPTR)

is the number of keys available in the master tables.

One last comment about the error handling. The key shortage error forces exits all the way out to CLI. You must restart ROOT.COM to load additional keys.

Assuming that the key supply drawdown for this distribution is acceptable, EXTRACT requires no additional operator action. It runs to completion and returns to the main ROOT menu.

VII. KEY DISTRIBUTION

The step after the key set extraction is the writing of the distribution tapes. This is initiated by the ROOT menu selection DISTRIB. Of the WCS this is the process that demands considerable operator attention. As a result, DISTRIB is quite verbose in providing instructions to the user.

After providing introductory information, DISTRIB determines the number of tapes that will be needed for this distribution. It then displays the count on the user's terminal and then enters a PAUSE state waiting for a command. When the operator has all the supplies ready, he returns a READY instruction and DISTRIB continues. If at this pause the decision is made to postpone this tape writing, an EXIT instruction returns to the ROOT main menu with no other action. The distribution can then be done at a later time.

Assuming the READY response is returned, DISTRIB.COM inquires for the identifier of the tape drive. The process defaults to the device name MTA0:. If this is correct, respond to the inquiry with <Return>. Otherwise, enter the correct identifier. If you do not know the correct identifier and happen to enter some junk, DISTRIB will accept it and then encounter a fatal error when it tries to mount the so-named device. The code is not resistant to user error. Learn this name if you do not know it.

Assuming the tape drive identifier is correct, DISTRIB requests the operator to load the tape and enter -READY- when the drive is on-line. The file transfer is automatic. The next message to the operator asks if he wants to delete the residual files. If the tape write went correctly, delete the files. If not, do not; they will come around again.

The last operator action is to unload the tape and label it with an external label (as in stick-on label). The process provides the operator ample node identification information to write on this label.

When the tape has been removed from the drive and labeled, the operator enters -READY- to cycle the tape write through the next set of files. The process concludes when it detects that no further files are remaining. Keep in mind that this is determined by the deletion of the residual files. If you do not delete these files eventually, the process will continue to loop ad nauseam.

At the conclusion of the DISTRIB.COM process, the operator has a tape for each of the WBCN nodes. These tapes are sent to the cognizant individual at each of the WBCN nodes for further processing. That moves the user's document to the site-specific part of the WCS operation.

VIII. THE SITE-SPECIFIC KEY MANAGEMENT

As was stated in the general comments, I assume that the site WBCN machines have been fully configured under VMS, that they have the INGRES DBMS installed and functioning properly, and that an account named CRYPTO has been created with the root directory [CRYPTO] located on the UD: logical device. Further, the CRYPTO account must be a valid INGRES user account. The software for this element has been made available on magnetic tape using the VMS Backup Utility.

The first step is to load the software from the tape to the VMS files. To do this create a subdirectory [.KEYUSER] in the crypto root directory. Use the backup command

```
$ BACKUP MFAØ: CRYPTO.BCK/SELECT = [CRYPTO.KEYUSER...] [CRYPTO.KEYUSER...]
```

This command transfers all the files and structure from the BACKUP saveset CRYPTO.BCK to the VMS directory [CRYPTO.KEYUSER]. Next, set the default directory for the process with the VMS command

```
$ SET DEFAULT [CRYPTO.KEYUSER]
```

Then issue the command

```
$ DIR
```

You should see the display on your screen that is shown in Fig. C-3.

```
$ dir

Directory UD:[CRYPTO.KEYUSER]

FETCHKEY.EXE;1      GETKEY.EXE;2      KEYLOAD.COM;18      LOGICALS.COM;4
SETUP.COM;5         SETUP.ING;4       TCOPY.ING;8

Total of 7 files.
```

Fig. C-3. VMS directory structure resulting from loading the save set CRYPTO.BCK on the WBCN gateway machine.

Assuming that everything checks out to this point, the next step is creating the database. This is accomplished by running SETUP.COM. You do this with the command

```
$ @ SETUP.COM
```

This process takes a little time. It is most helpful if you have a separate system manager account active on a separate terminal monitor at the same time. If you do, use the MONITOR utility to verify that SETUP.COM is proceeding and is not locked up because of some VMS/INGRES setup problems. If SETUP.COM does not complete successfully, get help to determine and resolve the problem. Then run SETUP.COM again.

Upon the successful completion of SETUP.COM, you should be able to verify the existence of the database named KEYUSER, using the CATALOGDB feature of INGRES. Further, you can verify the correct creation of the initial tables by entering the command

```
$ INGRES KEYUSER
```

This starts the basic INGRES process on the database KEYUSER. Enter help\g to the INGRES prompt (*). You should see displayed the same screen images as shown in Fig. C-4. You exit from the INGRES process by entering \q <Return>; this returns you to the DCL/CLI. When you are satisfied that SETUP.COM has run successfully, delete it from the system with the command

```
$ DELETE/CONF SETUP.COM
```

```
$ ingres keyuser
INGRES VAX Release 5.0/02a (vax.vms/01) login 13-FEB-1987 11:54:59
Copyright (c) 1986, Relational Technology Inc.

INGRES Release 5.0 (PRODUCTION)
go
* help
* \g
Executing . . .

name            owner       type        name            owner       type

nodetable       ingres      table       ptrtable        ingres      table
keytable        ingres      table       prdtable        ingres      table

continue
*
*
*
*
*
*
*
```

Fig. C-4. The database structure resulting from running SETUP.COM on the WBCN gateway.

IX. LOADING KEYS

After the database is set up, the next step is loading the keys from the distribution prepared on the WBCN KMM. This tape contains only the keys for this specific node. The keys are loaded by running the DCL command

```
$ KEYLOAD.COM
```

This process, after some initial information, queries the operator for the tape drive device identifier. The code defaults to a device identifier of MTA0:. If this is correct, then a <Return> is the correct response to the query. Otherwise, enter the correct tape drive device identifier.

The next step of the code asks the operator to load the tape on the designated drive. When the tape is loaded and the drive is on-line, the operator enters -READY-. Any other response causes the tape load request to loop. This is done intentionally. The tape contents are classified. When the decision is made to load the keys, it should be completed. After the tape is loaded and the READY response is made, the rest of the process is automatic. When the tape copy has ended, a message to that effect is posted to the terminal with the reminder to the operator to unload the tape.

The process is set to trap only one error. That error is a mismatch between the internal tape label and the WBCN node identifier. The code compares the response to the command

```
$ SHOW LOGICAL SYSNODE
```

with part of the internal tape label. If the two do not match, the whole key load sequence is skipped, an error message is sent to the terminal, and the process ends.

If this error occurs, I suggest you contact the central key distribution facility and prepare for a long discussion.

These two processes are the only ones needed for the local node key management.

X. THE CIPHER PROCESSES

After the keys are distributed, the last component of the WCS is the cipher processes. There are five subroutines used to supply the cipher processes: GETKEY, FETCHKEY, cbc_cipher, XOR, and PDES. The listings of these subroutines are given in Appendix B.

The subroutine GETKEY returns a key and a key pointer when called with the destination node designator. The pointer is the single piece of information that must be passed to the destination node so that it can identify the same key. It is the responsibility of the WCP to manage the exchange and retention of this information.

The subroutine FETCHKEY returns a key when it is called with a valid key pointer. If the pointer is not valid, FETCHKEY returns an error condition.

The last user interface subroutine is cbc_cipher. The details of its calling sequence are given in the listing in Appendix B.

Printed in the United States of America
Available from
National Technical Information Service
US Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Microfiche (A01)

Page Range	NTIS Price Code	Page Range	NTIS Price Code	Page Range	NTIS Price Code	Page Range	NTIS Price Code
001-025	A02	151-175	A08	301-325	A14	451-475	A20
026-050	A03	176-200	A09	326-350	A15	476-500	A21
051-075	A04	201-225	A10	351-375	A16	501-525	A22
076-100	A05	226-250	A11	376-400	A17	526-550	A23
101-125	A06	251-275	A12	401-425	A18	551-575	A24
126-150	A07	276-300	A13	426-450	A19	576-600	A25
						601-up*	A99

*Contact NTIS for a price quote.